The MagPi ESSENTIALS

CONQUER THE COMMAND LINE

The Raspberry Pi TERMINAL GUIDE

Written by **Richard Smedley**

TAKE US **ANYWHERE**

YOUR OFFICIAL RASPBERRY PI MAGAZINE
The MagPi
Win! 10 RasPi Model A+s
100 pages of hacking & making
TIME-LAPSE PHOTOGRAPHY
INTRODUCING GPIO ZERO
ASTRO PI UPDATE
30:00
MINUTE PROJECTS
Amazing ideas made in moments
3D PRINTED GAME BOY
PIXEL ART WITH THE SENSE HAT
BUILD A PINGOMETER
Also inside:
MAKE GAMES WITH RASPBERRY PI
MASTER PIMORONI'S SKYWRITER HAT
CREATE AMAZING MUSIC WITH SONIC PI
REVIEWED: SCRATCH, RASPBIAN & MORE
TOUCHSCREEN DISPLAY
THE ONLY MAGAZINE WRITTEN BY THE COMMUNITY, FOR THE COMMUNITY
NEW! YOUR OFFICIAL RASPBERRY PI MAGAZINE
The MagPi
Win! 5 worth £250
BUILD A
RASPBERRY PI ROBOT FOR £50

SAVE 45%
with a Newsstand subscription
(limited time offer)

FREE: ALL 30 LEGACY ISSUES NOW INCLUDED!

The MagPi Magazine

Available now
for smartphones & tablets

Available on the App Store

Get it on Google play

From just

£2.29 rolling subscription **or** **£19.99** full year subscription

Download the app – it's free!

- Get all 31 legacy issues free
- Instant downloads every month
- Fast rendering performance
- Live links & interactivity

WELCOME TO CONQUER THE COMMAND LINE

Sometimes only words will do. Graphic user interfaces (GUIs) were a great advance, creating an easy route into computer use for many non-technical users. For complex tasks, though, the interface can become a limitation: blocking off choices, and leaving a circuitous route even for only moderately complicated jobs.

(Re-)Enter the command line: the blinking cursor that many thought had faded away some time in the 1990s. For getting instructions from user to computer – in a clear, quick and unambiguous form – the command line is often the best way. It never disappeared on Unix systems, and now, thanks to Raspbian on the Raspberry Pi, a new generation are discovering the power of the command line to simplify complex tasks, or instantly carry out simple ones.

If you're not comfortable when faced with the $ prompt, then don't panic! We'll quickly make you feel at home, and able to find your way around the terminal on the Pi, or any other GNU/Linux computer: getting things done, and unlocking the power of the command line.

FIND US ONLINE raspberrypi.org/magpi **GET IN TOUCH** magpi@raspberrypi.org

The MagPi

Available on the App Store

Get it on Google play

EDITORIAL
Managing Editor: **Russell Barnes**
russell@raspberrypi.org
Technical Editor: **David Whale**
Sub Editors: **Laura Clay, Phil King, Lorna Lynch**

DESIGN
Critical Media: **criticalmedia.co.uk**
Head of Design: **Dougal Matthews**
Designers: **Lee Allen, Mike Kay**
Illustrator: **Sam Alder**

DISTRIBUTION
Seymour Distribution Ltd
2 East Poultry Ave,
London
EC1A 9PT | **+44 (0)207 429 4000**

SUBSCRIPTIONS
Select Publisher Services Ltd
PO Box 6337
Bournemouth
BH1 9EH | **+44 (0)1202 586 848**

recycle

In print, this product is made using paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

The MagPi magazine is published by Raspberry Pi (Trading) Ltd., Mount Pleasant House, Cambridge, CB3 0RN. The publisher, editor and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised in the magazine. Except where otherwise noted, content in this magazine is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). ISSN: 2051-9982.

The MagPi ESSENTIALS

CONTENTS

05 [CHAPTER ONE]
DON'T PANIC
Take a look around and discover things

10 [CHAPTER TWO]
READ/WRITE TEXT
Get working on files

15 [CHAPTER THREE]
PERMISSION TO INSTALL
Raspbian's system for installing and updating

20 [CHAPTER FOUR]
MANIPULATING TEXT
Connect together multiple simple commands

25 [CHAPTER FIVE]
CUSTOMISE THE COMMAND LINE
Make Raspbian a little more personal

30 [CHAPTER SIX]
CONNECTING DISKS
Tackle the management of removable storage

35 [CHAPTER SEVEN]
PREDICTABLE NETWORKING
Give the Pi a permanent network address of its own

40 [CHAPTER EIGHT]
COMMAND LINE PI
No need to turn it off and on again: just kill the process!

45 [CHAPTER NINE]
REMOTE PI
Accessing the Pi from remote PCs and devices with Secure Shell

50 [CHAPTER TEN]
DOWNLOADING & INSTALLING
Downloading and unpacking software, and creating new Raspbian SD cards

[RICHARD SMEDLEY]

Since soldering together his first computer - a ZX81 kit - and gaining an amateur radio licence as GW6PCB, Richard has fallen in and out of love with technology. Swapping the ZX81 for a guitar, and dropping ham radio for organic horticulture, he eventually returned to the command line, beginning with a computer to run his own business, and progressing to running all the computers of an international sustainability institution. Now he writes about Free Software and teaches edible landscaping.

The MagPi ESSENTIALS

[CHAPTER ONE]
DON'T PANIC

In the first chapter, we take a look around and discover that things aren't as strange as they might appear...

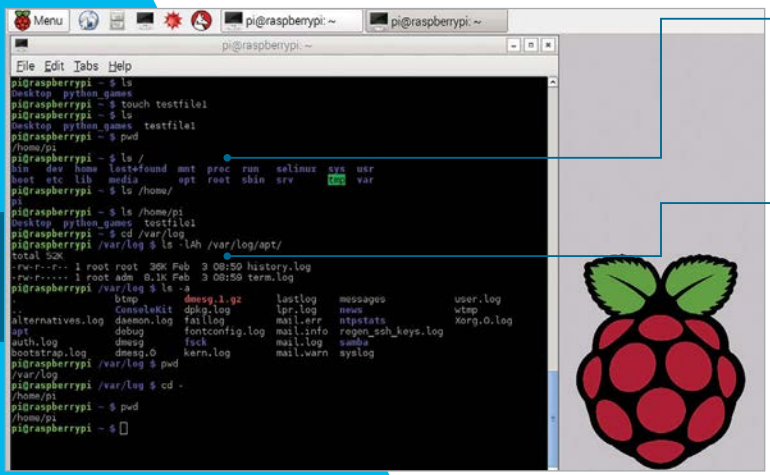
It's not a throwback to the past, but a quick and powerful way of getting your Pi to do what you want, without all that RSI-inducing menu chasing and icon clicking. The command-line interface was a great step up from manually toggling in your instructions in octal (base-8), using switches on the front of the machine! Graphical user interfaces (GUIs) brought friendly visual metaphor to the computer, losing some power and expressiveness. With the Pi, you can get the best of both worlds by knowing both: after reading through this guide, you'll soon be as comfortable at the command prompt as you are at your desktop.

When you boot up your Pi with Raspbian Wheezy installed, you arrive at the command line by default. You log in and type **startx** to get to the desktop. If you hold down the **ALT+CTRL** keys and press **F1** (the first function key on the keyboard), you'll see that the command line is still there. Press **ALT+F2** through to **F6** and you'll find five further virtual consoles waiting for you to log in.

You can drop into these any time you like, but for now press **ALT+F7** and you'll be back in mouse and menu land. The command line is also available through a program called a terminal emulator (often referred to as a term or xterm). You'll also find people referring to the shell, or Bash. Don't worry about that for now; just click on the icon at the top of the screen that looks like a black television screen,

[READ THE MANUAL]

Help is included, with man(ual) pages, but they can be a little overwhelming. Use them to check out some extra options beyond the switches like **-a** we used here. To read the ls man page, type **man ls**.



The command line is only a click away: it is called Terminal and you can find it under Accessories in the menu

Commands are terse, but, once learned, they're a quick way of navigating and reading your files and folders

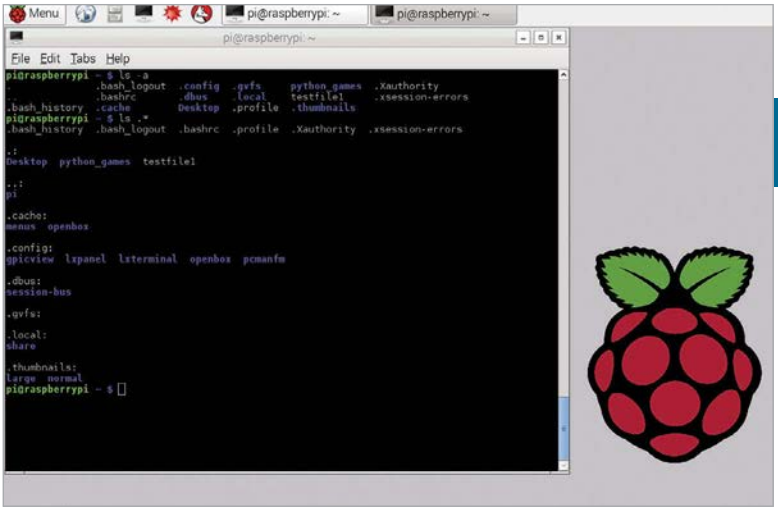


Fig 1 Switches modify behaviour in commands; ls -a shows (dot) files in your listing that are usually hidden from view

[PRESS RETURN]

To save repeating it in the text, we'll confirm here that each time you type in a command, you need to hit the **Return** key at the end, to tell the Pi you've issued Bash with a command.

or go to Accessories>Terminal in the menu: the terminal now awaits your commands.

Look around

If you're used to looking at files and folders in a file manager, try to clear your mind of the icons and concentrate on the names. Type **`ls`** and press **Return** (see the 'Press Return' boxout on this page). On a fresh Raspbian install you'll just see two directories: python_games and Desktop. Type **`ls python_games`** (also see the Lazy Completion boxout on page 8) and you should see a listing like **Fig 1**.

Commands like **`ls`** are not cryptic (at least not intentionally) but they are terse, dating back to a time when the connection to the computer was over a 110 baud serial line, from an ASR 33 teletype terminal. If you think it's strange to be defined by 50-year-old technology, just remember that your QWERTY keyboard layout was reputedly designed both to stop mechanical typewriter keys jamming, and to enable salespeople to quickly type 'typewriter' using the top row!

File path

You can list files and folders anywhere in your system (or other connected systems) by providing the path as an argument to your

command. The path is the folder hierarchy: on a Windows computer, in a graphical file browser, it starts with 'My Computer'; on your Pi it starts at `/`, pronounced 'root' when used on its own as the root of your filesystem. Try entering `ls /` – again we get terseness, and names like 'bin', which is short for binary, and is the directory where many programs are kept (enter `ls /bin` to see the details). `ls /dev` shows hardware devices in the Pi. Try `ls /home` – see that 'pi'? That's you: you are logged in as user pi. If you've changed your login name, or if you have created extra users, they'll all be listed there too: every user gets their own home directory; yours is the /home/pi folder in which we found ourselves in earlier. Before, with python_games, we used the relative path (the absolute path would be /home/pi/python_games) because we're already home. If you need to check your location, type `pwd` (present working directory).

> Commands are not cryptic (at least not intentionally), but they are terse

There's no place like ~

For any logged-in user, their home directory is abbreviated as `~` (the tilde character). Type `ls ~` and you'll see. There's apparently not much in your home directory yet, but Raspbian keeps a lot hidden from the casual glance: files and folders beginning with a dot, known as 'dot files', contain configuration information for your system and its programs. You don't need to see these files normally, but when you do, just ask `ls` to show you all files with a command switch. You can do this with either the full switch `--all`, or the abbreviation `-a` like so: `ls -a ~`. Traversing the pathways of the directory hierarchy can be easier from the command line than clicking up and down a directory tree, particularly with all the shortcuts given. Your `ls -a` showed you `.` and `..` as the first two directories; these shortcuts represent the current and the parent directory respectively. Try listing the parent directory – from /home/pi, `ls ../../` will show you two layers up. If you want to list the hidden files without the `.` and `..` appearing (after all, they're present in every directory, so you don't need to be told), then the switch to use is `-A`.

[LAZY COMPLETION]

You don't need to type all of `ls python_games` – after `ls p`, hit the **Tab** key and it will auto-complete. If you've more than one file beginning with p, they'll all be listed and you can type more letters and hit **Tab** again.

Before we move on to other commands, let's look briefly at chaining switches together: `ls -lAh ~`

`-l` gives you more information about the files and folders, and `-h` changes the units from bytes to KB, MB or GB as appropriate. We'll look at some of the extras the `-l` listing shows you in more detail later, particularly in chapters two and three.

Time for change

That's enough looking: let's start moving. `cd` is short for change directory, and moves you to anywhere you want in the filesystem: try `cd /var/log` and have a look (`ls`, remember). Files here are logs, or messages on the state of your system that are saved for analysis later. It's not something you'll often need to think about: Raspbian is a version of an operating system that also runs across data centres and supercomputers, where problem monitoring is very important. It is, however, useful to know, particularly if you have a problem and someone on a forum advises you to check your logs.

`cd ~` will take you where you expect it. Try it, then `pwd` to check. Now try `cd -` (that's a hyphen, or dash), the '`-`' is a shortcut for 'wherever I was before I came here'. Now we've looked around, we can move on to beginning to do things to our files.

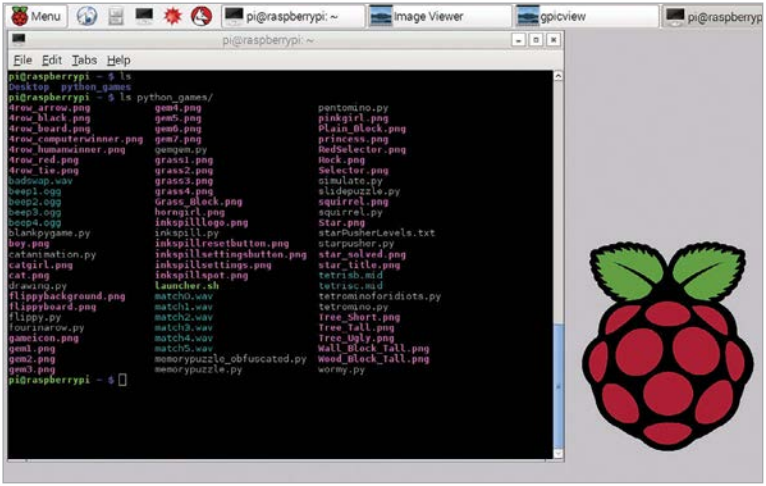
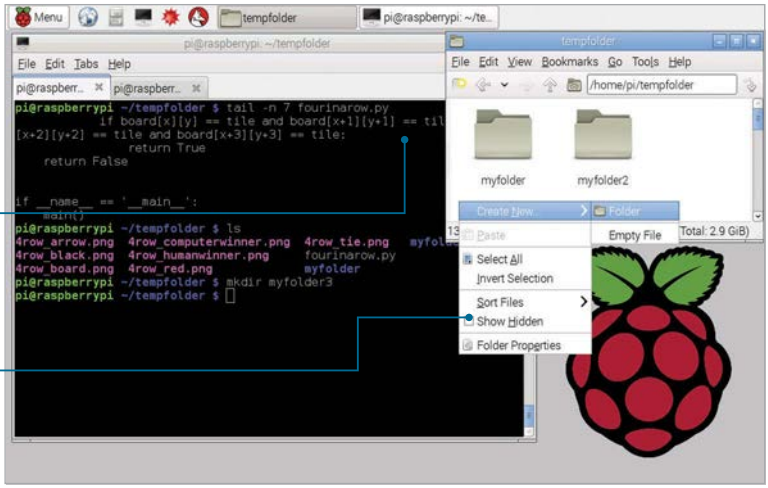


Fig 2 Who needs icons when you can fit a listing of 78 files into a small window? Coloured fonts indicate file types

The MagPi ESSENTIALS

[CHAPTER TWO] READ/WRITE TEXT

In this chapter, we get working on files.



The command line offers tools to get text from different parts of a file, like skipping to the conclusion

Create and name files and directories with keystrokes, rather than mouse-clicks and keystrokes

[MORE INFO]

Many utilities have info pages, giving far more information than their man page. If you're feeling brave, try **`info nano`** for a comprehensive guide to nano.

Now that we can navigate folders and list files, it's time to learn how to create, view, and alter both files and folders. Once more, it's not that the task is difficult, rather that the forms of the commands (particularly when editing) are unfamiliar, coming from an era before Common User Access (CUA) standards were created to ease switching between applications and operating systems.

Stick with it: with just the first two chapters of this book under your belt, you'll be able to do plenty of work at the command line, and start getting comfortable there.

Creating a directory

We're going to dive straight into working with files and folders by creating a new directory. Assuming you already have a terminal open (see 'Instant applications' box on page 13), and you're in your home directory (**`pwd`** to check, **`cd ~`** to get back there if necessary), type **`mkdir tempfolder`** and have a look with **`ls`**.

`mkdir`, as you've probably guessed, creates a new directory or folder. Let's use it to experiment with altering one of the included Python games. Don't worry: we're not going to be programming Python, just making a small change by way of illustration. **`cd tempfolder`** (use tab completion, **`cd t`** then hit the **`TAB`** key). We'll copy over the files from the games directory:

```
cp ../python_games/fourinarow.py .
cp ../python_games/4row_* .
```

Wildcard

You may remember that the `..` refers to the directory above. The `.` (dot) at the end of the commands refers to 'just here', which is where we want the files copied. Lastly, `4row_*` is read by the Pi as 'every file beginning `4row_`' – the `*` is known as a wildcard, and this one represents any number of characters (including none); there are other wildcards, including `?`, which means any single character.

Before we make any changes, try `python fourinarow.py` (after running `startx`) and you'll see you can run the local copy of the game (the Python Games part of the menu still reaches the original copy, of course). To change the game, we need an editor: there is a long and honourable tradition in the Unix and Linux world of having strong opinions on the merits or otherwise of various text editors – after all, they are an important tool for anyone who works daily with command scripts – but we'll sidestep the debate and use the Pi's built-in editor: nano. Unless you've previously used the Pico editor, which accompanied the Pine email client on many university terminals in the 1980s and 1990s, it will seem a little odd (**Fig 1**). That's because its conventions predate the **CTRL+C** for copy type standards found in most modern programs. Bear with us.

[SWITCHING HELP]

You don't need to wade through the man page to see what switches are available: type `--help` after the command to be shown your options, e.g. `rm --help`

> "We're going to dive straight into working with files and folders by creating a new directory."

Editing and paging

`nano fourinarow.py` will open the game for editing; use the arrow keys to go down nine lines, and along to the `BOARDHEIGHT` value of `6`. Change it to `10` (both the **BACKSPACE** and **DELETE** keys will work in nano). The last two lines of the screen show some shortcuts, with

[INSTANT APPLICATIONS]

Although you can open the terminal emulator from the menu – Accessories > Terminal – for this, and any other app, just hit **ALT+F2** and type its command name: `lxterminal`.

^ (the caret symbol) representing the **CTRL** key: **CTRL+O**, followed by **RETURN** will 'write out' (save) the file; then use **CTRL+X** to exit. Now, `python fourinarow.py` will open an oversize board, giving you more time to beat the computer, should you need it. However, there's now no room to drag the token over the top of the board: go back and change the `BOARDHEIGHT` value to `9`, with nano.

If you want to take a look through the `fourinarow.py` listing without entering the strange environment of nano, you can see the entire text of any file using `cat`: eg `cat fourinarow.py`. Unfortunately, a big file quickly scrolls off the screen; to look through a page at a time, you need a 'pager' program. `less fourinarow.py` will let you scroll up and down through the text with the **PAGE UP** and **PAGE DOWN** keys. Other keys will do the same job, but we'll leave you to discover these yourself. To exit `less`, hit **Q** (this also works from man and info pages, which use a pager to display text).

Cats, heads & tails

If editor wars are a Unix tradition we can safely ignore, there's no getting away from another tradition: bad puns. `less` is an improvement over `more`, a simple pager; the respective man pages will show you the differences. One advantage the relatively primitive `more` has is that at the end of a file it exits automatically, saving you reaching for the **Q** button. Admittedly, this is not a huge advantage, and you can always use `cat`.

Fortunately, `cat` is not a feline-based pun, but simply short for 'concatenate': use it with more than one file and it concatenates them together. Used with no argument – type `cat` – it echoes back what you type after each **ENTER**. Hit **CTRL+C** to get out of this when you've finished typing in silly words to try it. And remember that **CTRL+C** shortcut: it closes most command-line programs, in the same way that **ALT+F4** closes most windowed programs.

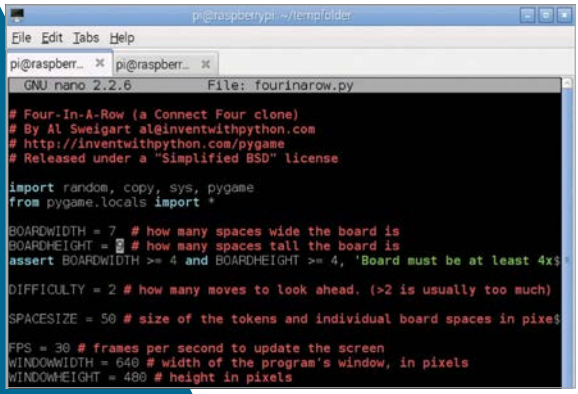

pi@raspberrypi: ~/templfolder
File Edit Tabs Help
pi@raspberr...
pi@raspberr...
GNU nano 2.2.6 File: fourinarow.py
# Four-In-A-Row (a Connect Four clone)
# By Al Sweigart al@inventwithpython.com
# http://inventwithpython.com/pygame
# Released under a "Simplified BSD" license
import random, copy, sys, pygame
from pygame.locals import *
BOARDWIDTH = 7 # how many spaces wide the board is
BOARDHEIGHT = 6 # how many spaces tall the board is
assert BOARDWIDTH >= 4 and BOARDHEIGHT >= 4, 'Board must be at least 4x4'
DIFFICULTY = 2 # how many moves to look ahead. (>2 is usually too much)
SPACESIZE = 50 # size of the tokens and individual board spaces in pixels
FPS = 30 # frames per second to update the screen
WINDOWWIDTH = 640 # width of the program's window, in pixels
WINDOWHEIGHT = 480 # height in pixels

Fig 1 The default editor, Nano, has unusual command shortcuts, but they're worth learning, as you'll find Nano installed on virtually all Linux boxes, such as your web host

You can peek at the first or last few lines of a text file with **head** and **tail** commands. **head fourinarow.py** will show you the first ten lines of the file. **head -n 5 fourinarow.py** shows just five lines, as does **tail -n 5 fourinarow.py** with the last five lines. On the Pi, **head -5 fourinarow.py** will also work.

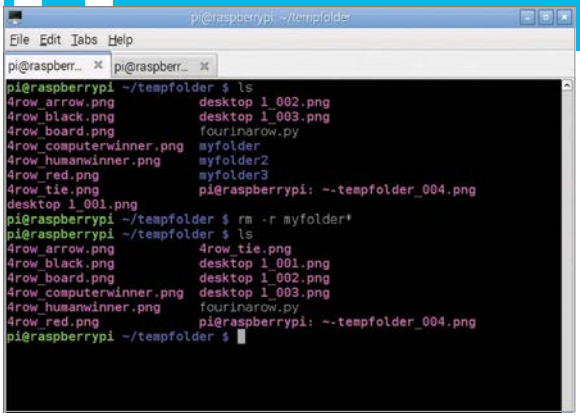


Fig 2 rm is a powerful removal tool: use with great care!

Remove with care

nano afile.txt will create a new file if afile.txt does not already exist: try it, and see if it works when you exit the file before writing and saving anything. We've done a lot already (at least, nano makes it feel like a lot), but it's never too early to learn how to clean up after ourselves. We'll remove the files we've created with **rm**. The remove tool should always be used with care: it has some built-in safeguards, but even these are easy to override (**Fig 2**). In particular, *never* let anyone persuade you to type **rm -rf /** – this will delete the entire contents of your Pi, all the programs, everything, with little to no chance of recovery.

Have a look at what files we have: if you're still in the tempfolder/ you made, then **ls** will show you the Four-in-a-Row files you copied here. Remove the program, then the .png files with careful use of the ***** wildcard.

```
rm fourinarow.py
rm 4row_*.png
```

cd .. to get back to /home/pi and **rm -r tempfolder** will remove the now empty folder. The **-r** (recursive) option is necessary for directories, and will also remove the contents if any remain.

In the next chapter, we'll delve into file permissions and updating your Pi's software from the command line.

The MagPi ESSENTIALS

[CHAPTER THREE]
PERMISSION TO INSTALL

In chapter three, we look at Raspbian's efficient system for installing and updating software, among other things.

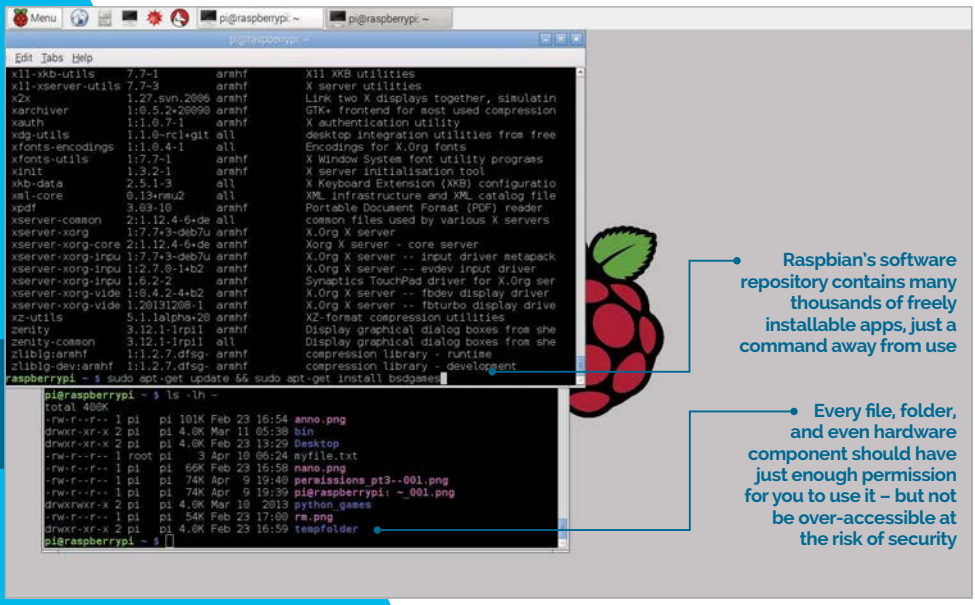
Installing software should be easy, but behind every piece of software is an evolving set of dependencies that also need installing and updating. Keeping them separate reduces unnecessary bloat and duplication, but adds the potential for bugs, missing files, and even totally unresolvable clashes.

Fortunately, Debian GNU/Linux cracked the problem back in the 1990s with the Debian Package Management system and the Advanced Package Tool (APT), and Debian-based systems, like Ubuntu and the Pi's Raspbian, inherit all of the benefits. Here we'll show you the basics you need to know to install new software and keep your system up to date from the command line, and then look at the not entirely unrelated field of file ownership and permissions.

Using the **apt** command to update your system's list of installable software should be as simple as issuing the command like so: **apt-get update**. Try this logged in as user pi, though, and you'll just get error messages. The reason for this is that changing system software on a GNU/Linux (or any type of Unix) system is a task restricted to those with administrative permissions: the godlike superuser, or admin, also known as root.

[SHARED RESPONSIBILITY]

If you share your Pi, it's worth reading up on sudo, and the visudo command to find how to give limited but useful admin privileges to the other users.


Raspbian's software repository contains many thousands of freely installable apps, just a command away from use
Every file, folder, and even hardware component should have just enough permission for you to use it – but not be over-accessible at the risk of security

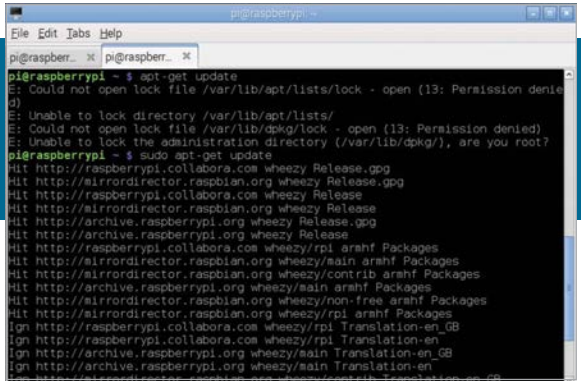

File Edit Tabs Help
pi@raspberr... pi@raspberr...
pi@raspberrypi ~ $ apt-get update
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
E: Could not open lock file /var/lib/dpkg/lock - open (13: Permission denied)
E: Unable to lock the administration directory (/var/lib/dpkg/), are you root?
pi@raspberrypi ~ $ sudo apt-get update
Hit http://raspberrypi.collabora.com wheezy Release.gpg
Hit http://mirrordirector.raspbian.org wheezy Release.gpg
Hit http://raspberrypi.collabora.com wheezy Release
Hit http://mirrordirector.raspbian.org wheezy Release
Hit http://archive.raspberrypi.org wheezy Release.gpg
Hit http://archive.raspberrypi.org wheezy Release
Hit http://raspberrypi.collabora.com wheezy/rpi armhf Packages
Hit http://mirrordirector.raspbian.org wheezy/main armhf Packages
Hit http://mirrordirector.raspbian.org wheezy/contrib armhf Packages
Hit http://archive.raspberrypi.org wheezy/main armhf Packages
Hit http://mirrordirector.raspbian.org wheezy/non-free armhf Packages
Hit http://mirrordirector.raspbian.org wheezy/rpi armhf Packages
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-en_GB
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-en
Ign http://archive.raspberrypi.org wheezy/main Translation-en_GB
Ign http://archive.raspberrypi.org wheezy/main Translation-en

Fig 1 Raspbian updates its listing of thousands of available apps, providing you give it admin permissions

Pseudo root, su do

We'll get onto permissions properly a bit later, but for now you'll be pleased to know that you can fake it, using the **sudo** command. **sudo** potentially offers a fine-grained choice of permissions for users and groups to access portions of the admin user's powers. However, on the Pi, Raspbian assumes, quite rightly, that the default user will be someone just wanting to get on with things, and **sudo** in front of a command will pretty much let you do anything.You have been warned!

The following two commands will update Raspbian's installed software (**Fig 1**):

```
sudo apt-get update
sudo apt-get upgrade
```

You can wait for one to finish, check everything is okay, then issue the other command, or you can save waiting and enter both together with:

```
sudo apt-get update && sudo apt-get upgrade
```

The **&&** is a Boolean (logical) AND, so if the first command doesn't run properly, the second one will not run at all. This is because for a logical AND to be true, both of its conditions must be true.

It's always worth running the **update** command before installing new software too, as minor updates are made in even stable distributions: should a security problem be found in any of Raspbian's

software. We've just run an update, so no need to repeat that for now. Sticking with a command-line theme, we're going to install an old suite of terminal games:

```
sudo apt-get install bsdgames
```

Searchable list

It is possible to find particular apps with apt-cache search: **apt-cache search games**. You can also examine individual packages with apt-cache show: **apt-cache show bsdgames**.

Apt is actually a front end to the lower-level **dpkg**, which you can call to see what you have installed on the system: **dpkg -l**. Even on a fresh system, that's a large listing: we'll show you how to get useful information from such listings another time.

Downloaded packages hang around in **/var/cache/apt** and if you find yourself short on disk space, issuing **sudo apt-get clean** will clear out the archive, without affecting the installed software.

> "It's always worth running the update command before installing new software..."

Now, remember the extra details that **ls -lh** showed us in part 1? Try **ls -lh /etc/apt**.

That **-rw-rw-r--** at the beginning of the listing for **sources.list** comprises file attributes, telling you who may use the file. Other entries in the listing have a **d** at the beginning, indicating they are directories. You'll also see hardware devices have a **c** here, for character device – **ls -l** on **/dev/input**, for example. On Linux, everything is a file, even your mouse! An **a** tells us this is just a regular file; it's the remaining nine characters of the group that cover permissions.

Every file has an owner and a group membership. Files in your home directory belong to you. If you're logged in as user pi and **ls ~ -l**, you'll see **pi pi** in each listing, telling you the owner and the group.

[FREE TO USE]

Software in the main repository is not just free to use, but freely modifiable and redistributable too. Free software, like Raspbian's Debian base, is built on sharing: for education and for building community.

[PROBLEMS?]

Fine-grained permissions make for greater security, but can trip you up. Typing **sudo** in front of a command that doesn't work is both a diagnosis and a quick workaround of a permissions problem.

Note that we put the switch at the end this time: that's a bad habit under certain circumstances, but we're just showing you what's possible. Owner and group aren't always the same, as `ls -l /dev` will show you.

Octal version

Those numbers are an octal representation of user, group, and others' permissions: in each case, read is represented by 4, write by 2, and execute by 1, all added together. So here we have 7s for read+write+execute for user and group, and 5 for read+execute for all other users. `ls -l` and you'll see we're back to `-rwxrwxr-x`.

You can use `chown` to change who owns a file and `chgrp` to change which group it belongs to. Make a new text file and `sudo chown root myfile.txt` – now try editing it and you'll find that while you can read the file, you can no longer write to it. You can also make a file that you can write to and run, but not read!

In the next chapter, we'll be doing useful things with the output of our commands; before moving on, though, why not try your hand at **robots** from the **bsdgames** package we installed? It will come in handy later for another purpose.

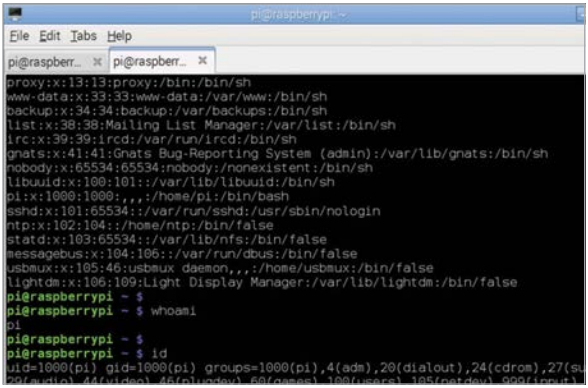
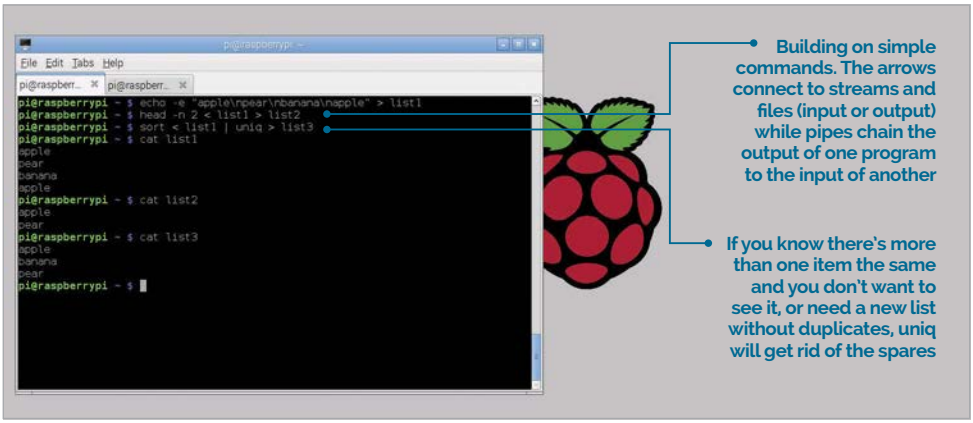


Fig 2 Raspbian tells you who you are, and what group access you have, for permission to use and alter files and devices

The MagPi ESSENTIALS

[CHAPTER FOUR]
MANIPULATING TEXT

Discover pipes and learn how to connect multiple simple commands together for more powerful text processing.


File Edit Tabs Help
pi@raspberr... pi@raspberr...
pi@raspberrypi ~ $ echo -e "apple\npear\nbanana\napple" > list1
pi@raspberrypi ~ $ head -n 2 < list1 > list2
pi@raspberrypi ~ $ sort < list1 | uniq > list3
pi@raspberrypi ~ $ cat list1
apple
pear
banana
apple
pi@raspberrypi ~ $ cat list2
apple
pear
pi@raspberrypi ~ $ cat list3
apple
banana
pear
pi@raspberrypi ~ $
Building on simple commands. The arrows connect to streams and files (input or output) while pipes chain the output of one program to the input of another
If you know there's more than one item the same and you don't want to see it, or need a new list without duplicates, uniq will get rid of the spares

[ABSOLUTE PATH]

We're using `~/mylisting4.txt` with `~` short for `/home/pi`. If you `cd` to `~` then you can simply use the filename without the `~/`

The Unix family of operating systems, which includes other flavours of GNU/Linux and also Apple's Mac OS X, deals with data from commands as streams of text. This means that commands can be chained together in countless useful ways. For now, though, we'll focus on giving you a firm foundation to building your own custom commands.

Getting our feet wet

When a command is called at the terminal, it is given three streams, known as standard input (stdin), standard output (stdout), and standard error (stderr). These streams are plain text, and treated by the Pi as special files. As we noted in chapter 3, 'everything is a file': this is what gives the Pi and other Unix family systems the ability to put together simple commands and programs to build complex but reliable systems.

Normally, stdin is what you enter into the terminal, while stdout (command output) and stderr (any error messages) appear together. The reason the last two have a separate existence is that you may want to redirect one of them – error messages, for example – somewhere away from the regular output your commands produce. We'll look at separate error messages later, but first we need to know how to redirect and connect our output to other commands or files.

Connecting commands together are pipes, the '|' symbol found above the backslash on both GB and US keyboards (although the two

keyboards for English speakers place the \ respectively to the left of Z, and at the far right of the home row). When you type a command such as **ls -l**, the output is sent by Raspbian to the stdout stream, which by default is shown in your terminal. Adding a pipe connects that output to the input (stdin stream) of the next command you type. So…

```
ls -l /usr/bin | wc -l
```

…will pass the long listing of the **/usr/bin** directory to the wordcount (**wc**) program which, called with the **-l** (line) option, will tell you how many lines of output **ls** has. In other words, it's a way of counting how many files and folders are in a particular directory.

Search with grep

One of the most useful commands to pass output to is **grep**, which searches for words (or Regular Expressions, which are powerful search patterns understood by a number of commands and languages), like so:

```
grep if ~/python_games/catanimation.py
```

This displays every line in the **catanimation.py** file containing the character sequence 'if' (**Fig 1**)– in other words not just the word 'if', but words like 'elif' (Python's **else if**), and words like 'gift' if they were present. You can use Regular Expressions to just find lines with 'if', or lines beginning with 'if', for example.

Piping search results and listings to **grep** is the way we find a needle in one of Pi's haystacks. Remember **dpkg** from the last chapter, to see what was installed? Try…

```
dpkg -l | grep -i game
```

…to remind yourself which games you've installed (or are already installed). The **-i** switch makes the search case insensitive, as the program may be a 'Game' or 'game' in the description column. A simple **dpkg -l | more** lets you see output a page at a time.

sort will, as the name suggests, sort a listing into order, with various tweaks available such as **-f** to bring upper and lower case together.

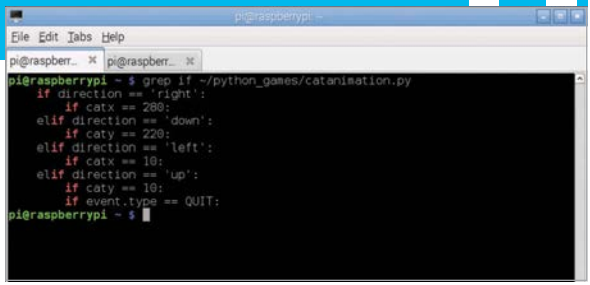


Fig 1 No matter how long the file, `grep` will dig out the lines you need . It's also handy for finding the results you want from a multi-page output

[FILING HOMEWORK]

There are many more commands beyond `grep`, `sort` and `uniq` that can be chained together. Take a look at `cut` if you're feeling adventurous.

One way to collect unsorted data is to combine lists. **`sort`** will put the combined listing back in alphabetical order:

```
ls ~ ~/python_games | sort -f
```

Suppose you copied one of the games to your home directory to modify: you know it's there, but you don't want to see the same name twice in the listings. **`uniq`** will omit the duplicated lines or, with the **`-d`** switch, show only those duplicates.

```
ls ~ ~/python_games | sort  -f| uniq
```

File it away

Pipes are not the only form of redirection. **`>`** (the 'greater than' symbol) sends the output of a program into a text file, either creating that text file in the process, or writing over the contents of an existing one.

```
ls /usr/bin > ~/mylisting4.txt
```

Now look in **mylisting4.txt** and you'll see the output of **`ls /usr/bin`**. Note that each item is on a separate line (**Fig 2**). Your terminal displays multiple listings per line for space efficiency; however, for easy compatibility between commands, one listing per line is used. Most commands operate on lines of text: for example, **`grep`** showed you in which lines it found 'if'. Note that some commands need a dash as a placeholder for the stdin stream being piped to them:

```
echo "zzzz is not a real program here" | cat mylisting4.txt -
```

Appending

If you want to add something to the end of a file without overwriting the contents, you need **>>**.

```
echo "& one more for luck!" >> ~/mylisting4.txt
```

echo simply displays whatever is in the quote marks to stdout; the **-e** switch lets you add in special characters, like \n for newline (see below). Remember, you can look at the last few lines of a file with **tail ~/mylisting4.txt. <** will link a program's input stream to the contents of a file or stream. Make an unsorted list to work on, and **sort** it:

```
echo -e "aardvark\nplatypus\njellyfish\
naardvark" > list1
sort < list1
```

You can also combine **<** and **>**:

```
head -n 2 < list1 > list2
```

...will read from **list1**, passing it to **head** to take the first two lines, then putting these in a file called **list2**. Add in a pipe:

```
sort < list1 | uniq > list3
```

Lastly, let's separate that stderr stream: it has file descriptor 2 (don't worry too much about this), and **2>** sends the error messages to any file you choose:

```
  cat list1 list2 list3
list42 2>errors.txt
```

The screen will display the 'list' files you do have, and the 'No such file or directory' message(s) will end up in **errors.txt** – **2>>** will append the messages to the file without overwriting previous contents.

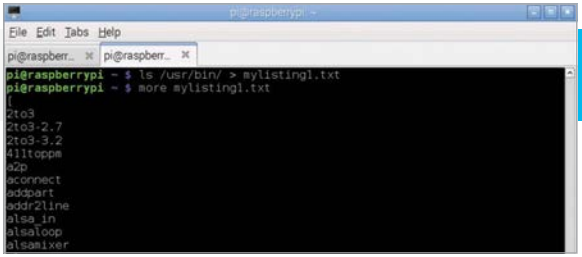
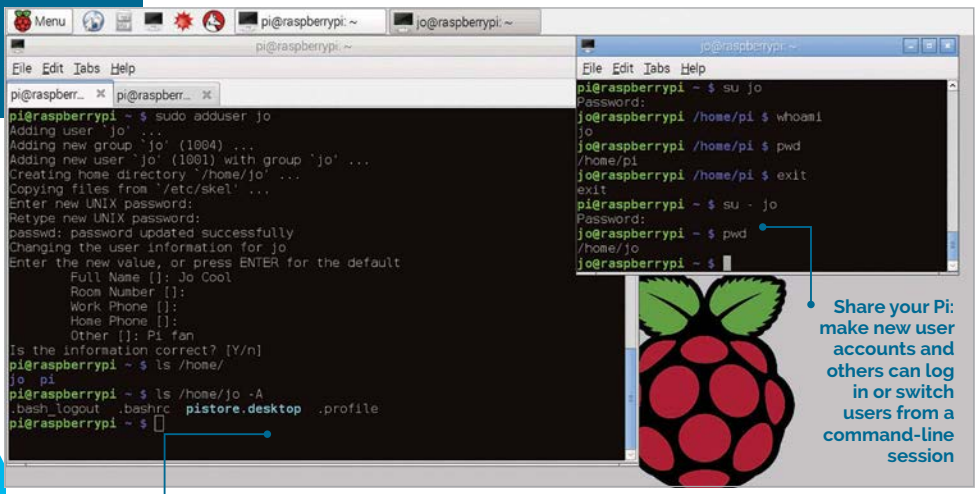


Fig 2 With redirection, you can get all of the output from a command saved straight into a text file. Save your error messages to ask about them on the forums!

The MagPi ESSENTIALS

[CHAPTER FIVE]
CUSTOMISE THE COMMAND LINE

In this chapter, we make Raspbian a little more personal as we get it to behave and look just the way we want it to.



Share your Pi: make new user accounts and others can log in or switch users from a command-line session

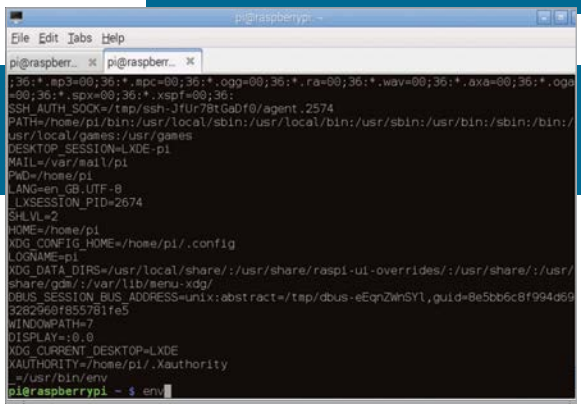
The command-line environment is personal to each user. You can change your identity with or without a change of environment, depending upon what you need to do in another role

Take a look at that blinking cursor on your terminal, and at what's behind it: **pi@raspberrypi ~ $**

The $ is known as the 'dollar prompt', awaiting your command; before it you see the **~** (tilde), shorthand for 'home' – which is **/home/pi** in this case. Before that is [user name]@[computer name], in the form **pi@raspberrypi**. Not only is this informative (at least if you've forgotten who and where you are), but it's also something you can change and personalise.

New user

Let's start with that user name: pi. If more than one person in your family uses the Pi, you may want to keep the pi user for shared projects, but set up individual login accounts for family members, including yourself. Creating a new user in Raspbian is easy: **sudo adduser jo** ...will create a new user account named **jo**. You will be prompted for a password (pick a good one) and lots of irrelevant info (dating back to shared university computers of the 1970s) that you can safely ignore by just pressing **ENTER** at each prompt. Now we have a user account for jo, have a look at **/home/jo**. Does it look empty? Use **ls -A**. Jo has never logged into the computer, so you will see the absence of most of the contents of /home/pi for now, such as **~/.gconf**, but there is a **.bashrc** and a couple of other config files.



Above Bash stores information, from your previous 'present working directory' to who you are, in environmental variables like OLDPWD and USER. See individual variables with e.g. echo $USER, or view them all with env

Not every user has a home directory and logs in: **`cat /etc/passwd`** ...and you'll see a lot of users listed that aren't people. This is because files and programs running on a Unix-type system have to belong to a user (and a group – take a look at **`/etc/group`**), as we saw back in chapter 1 when we did **`ls -l`**. The user passwords are fortunately not listed in the **`/etc/passwd`** file in plain text, so if you want to change a password you'll need to use the **`passwd`** command: **`sudo passwd jo`** will change the password for user jo. If you're logged in as user pi, then simply calling **`passwd`** will prompt you to change pi's password.

Transformations in the virtual world are always easier than those in nature, and this is the case with switching from being 'pi' to 'jo': we use the change (or substitute) user command, **`su`**, like so: **`su jo`**. After typing this, you should see the prompt change to **`jo@raspberry`**; you can also confirm who you are logged in as with **`whoami`**.

[HOME RUN]

If you're logged in as user **pi**, then **~** is a shortcut to **/home/pi** – but **`ls ~jo`** can be used as a shortcut to list **/home/jo**, substituting any other user name as desired, with tab completion working after **`~j`** is typed.

Changing identity

`su - jo` (note the dash) is usually preferred, as you'll gain all of jo's specific environment settings, including placing you in **/home/jo**. Note that on many other Linux systems, **`su`** on its own will enable you to become the root or superuser, with absolute powers (permissions to run, edit, or delete anything). Raspbian (and some other popular GNU/Linux systems like Ubuntu) prefer **`sudo`** to run individual programs with root permissions. Root's godlike powers may be temporarily attained with **`sudo -s`** – try it and note how the prompt changes – but it's generally a bad idea to run with more permissions than you need, for the same reason it's a bad idea to run with scissors! For any user, you can customise elements of their command-line

use most simply by editing **~/.bashrc**. Take a look through that configuration file now: **more ~/.bashrc**. Note a number of variables in all capital letters, such as **PATH**, **HISTSIZE**, and **PS1**. The last of these controls the prompt you see, currently **jo@raspberry ~ $**. To change it (for the duration of your current terminal session), try something like: **export PS1="tutorial@magpi > "**

This is a temporary change: type **exit** and you've left the **su** value of **jo**, so you'll see **pi@raspberry ~ $** once more. If you **su** back to jo, the magpi prompt will still be gone. To make your change permanent, you need to put the **PS1** value you want into **~/.bashrc**. A search around the web will bring up many fancy options for better customising the Bash prompt.

The **~/.bashrc** file is read upon each login to a Bash session, or in other words, every time you log into a console or open a terminal. That's unless you change Raspbian's default shell away from Bash,

> "Transformations in the virtual world are always easier than those in nature..."

something you may have reason to do in the future – there are interesting alternatives available for extra features or for smaller memory footprint – but let's not worry about that for now. You can put all sorts of commands in there to personalise your environment: command aliases are great for regularly used combinations.

Alias

See what's already there with: **grep alias ~/.bashrc**

There are a few already in there, particularly for the **ls** command. One entry is: **# alias ll='ls -l'**. This sounds quite useful, although the **#** indicates that it is 'commented out', which means that it will not be read by Bash. Open **.bashrc** in your text editor; the simple text editor from the Accessories menu will do for now, as although we've touched on using nano for editing text from the command line, we aren't going to go into this in detail until the next chapter. Removing the # will

[BASIC ACCOUNT]

adduser creates a new user, then takes care of all of the extra details like making a home directory. If all you want is a user created with no extra frills, then the command you want is **useradd**.

[WHO AM I?]

From a virtual console (before typing `startx`), `su` and that's who you're logged in as. From an xterm, you can change to someone else, but start another app from the menu and you'll be back to your original login.

mean that now when you type `ll`, you'll get the action of running `ls -l`. Handy, but we could make it better. Change it to: `alias ll='ls -lAhF'`...and you'll get an output in KB or MB, rather than bytes, along with trailing slashes on directory names and the omission of the ever present `.` and `..` (current and parent) directories. Changes take effect after you next start a Bash session, but you can just run that alias as a command (**Fig 1**). To disable an alias for a session, use: `unalias ll`

Key point

We'll end with the very first thing many users need to change: the keyboard map. The system-wide setting is in **/etc/default/keyboard**, but often you need to change it just for individual users. If £ signs and letters without accents are not sufficient for them, log in as the user who wants a different keyboard, or add **sudo** and the correct path to the commands below. For example, for a Greek keyboard:

```
touch ~/.xsessionrc
echo "setxkbmap el" > ~/.xsessionrc
```

Replace `el` with `pt`, `us`, or whatever language you desire. Note that config file we created – **.xsessionrc** – it holds settings that are read when we start the GUI with `startx`, so the keyboard setting will cover not just the terminal, but every app used in the session.

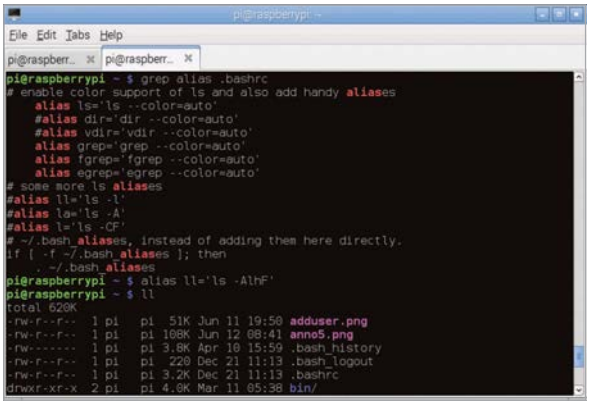
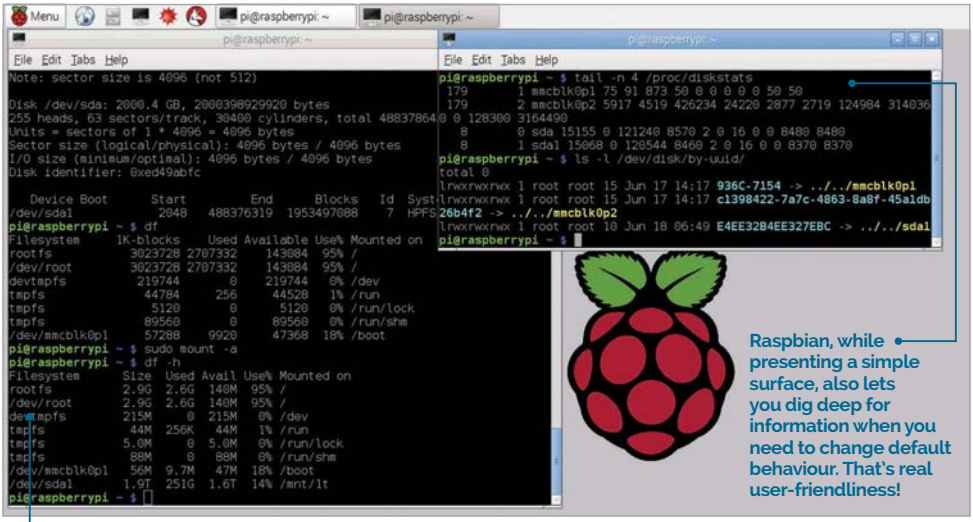

File Edit Tabs Help
pi@raspberr... pi@raspberr...
pi@raspberrypi ~ $ grep alias .bashrc
enable color support of ls and also add handy aliases
alias ls='ls --color=auto'
#alias dir='dir --color=auto'
#alias vdir='vdir --color=auto'
alias grep='grep --color=auto'
alias fgrep='fgrep --color=auto'
alias egrep='egrep --color=auto'
some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'
# ~/.bash_aliases, instead of adding them here directly.
if [-f ~/.bash_aliases]; then
. ~/.bash_aliases
pi@raspberrypi ~ $ alias ll='ls -AlhF'
pi@raspberrypi ~ $ ll
total 620K
-rw-r--r-- 1 pi pi 51K Jun 11 19:50 adduser.png
-rw-r--r-- 1 pi pi 108K Jun 12 08:41 anno5.png
-rw------- 1 pi pi 3.8K Apr 10 15:59 .bash_history
-rw-r--r-- 1 pi pi 220 Dec 21 11:13 .bash_logout
-rw-r--r-- 1 pi pi 3.2K Dec 21 11:13 .bashrc
drwxr-xr-x 2 pi pi 4.0K Mar 11 05:38 bin/

Fig 1 Those terse, two- or three-letter commands are not set in stone: make your own shortcuts to keep, or just for use over a specific session

The MagPi ESSENTIALS

[CHAPTER SIX] CONNECTING DISKS

For chapter six, we're tackling the management of removable storage.


Menu
pi@raspberrypi: ~
File Edit Tabs Help
Note: sector size is 4096 (not 512)
Disk /dev/sda: 2000.4 GB, 2000398929920 bytes
255 heads, 63 sectors/track, 30400 cylinders, total 488378864
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0xed49abfc
Device Boot Start End Blocks Id System
/dev/sda1 2048 488376319 1953497088 7 HPFS
pi@raspberrypi ~ $ df
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 3023728 2707332 143084 95% /
/dev/root 3023728 2707332 143084 95% /
devtmpfs 219744 0 219744 0% /dev
tmpfs 44784 256 44528 1% /run
tmpfs 5120 0 5120 0% /run/lock
tmpfs 89560 0 89560 0% /run/shm
/dev/mmcblk0p1 57288 9920 47368 18% /boot
pi@raspberrypi ~ $ sudo mount -a
pi@raspberrypi ~ $ df -h
Filesystem Size Used Avail Use% Mounted on
rootfs 2.9G 2.6G 140M 95% /
/dev/root 2.9G 2.6G 140M 95% /
devtmpfs 215M 0 215M 0% /dev
tmpfs 44M 256K 44M 1% /run
tmpfs 5.0M 0 5.0M 0% /run/lock
tmpfs 88M 0 88M 0% /run/shm
/dev/mmcblk0p1 56M 9.7M 47M 18% /boot
/dev/sda1 1.9T 251G 1.6T 14% /mnt/1t
pi@raspberrypi ~ $
File Edit Tabs Help
pi@raspberrypi ~ $ tail -n 4 /proc/diskstats
179 1 mmcblk0p1 75 91 873 50 0 0 0 0 50 50
179 2 mmcblk0p2 5917 4519 426234 24220 2877 2719 124984 314036
0 0 128300 3164490
8 0 sda 15155 0 121240 8570 2 0 16 0 0 8480 8480
8 1 sda1 15068 0 120544 8460 2 0 16 0 0 8370 8370
pi@raspberrypi ~ $ ls -l /dev/disk/by-uuid/
total 0
lrwxrwxrwx 1 root root 15 Jun 17 14:17 936C-7154 -> ../../mmcblk0p1
lrwxrwxrwx 1 root root 15 Jun 17 14:17 c1398422-7a7c-4863-8a8f-45a1db26b4f2 -> ../../mmcblk0p2
lrwxrwxrwx 1 root root 10 Jun 18 06:49 E4EE32B4EE327EBC -> ../../sda1
pi@raspberrypi ~ $

Raspbian, while presenting a simple surface, also lets you dig deep for information when you need to change default behaviour. That's real user-friendliness!

Even simple utilities have multiple uses: df, by showing space available, reminds the user which disks are mounted and can be accessed by the Pi

Although Raspbian Wheezy will, when booted as far as the GUI, automatically mount any disk-type device (USB flash key, camera, etc.) plugged into the USB port and offer to open it for you (**Fig 1**), you may wish to get more direct control of the process. Or, as is more often the case, you may want to mount a disk when the Raspberry Pi is running a project that doesn't involve anyone getting as far as typing **`startx`** at the command line, as such graphical fripperies aren't necessary for most connected devices.

[IN DEPTH]

If you want to delve deeper into what goes on inside Raspbian and other GNU/Linux systems, try Brian Ward's *How Linux Works*, which we reviewed in *The MagPi #32*.

Connected or mounted?

Plugging a drive or flash memory device into your Pi (*connecting* it to your computer) is not the same as making it available for the Pi to interact with (*mounting* it) so that Raspbian knows what's on it and can read, write, and alter files there. It's an odd concept to accept: the computer knows there's a disk plugged in, but its contents remain invisible until the Pi is told to mount it. It's a bit like seeing a book on your shelf, but not being allowed to open or read it.

Disks and disk-like devices are mounted by Raspbian on a virtual file system, and you'll rarely need to worry about what goes on beneath that layer of abstraction, but to see some of it, type **`mount`**. The information displayed is of the form *device* on *mount point*, *file-system*

type, *options*. You'll see lots of device 'none' for parts of the virtual system that you don't need to worry about; the devices that concern us start with **/dev/** and have names like **/dev/mmcblk0p1** for partitions of the Pi's SD card, and **/dev/sda1** for plugged-in USB drives.

Plug in a USB drive (remember that the Pi is not happy to power drives itself: either use a powered drive, or plug a USB flash drive into a powered USB hub). If you haven't yet typed **`startx`**, then the disk will not get automatically mounted; if you have, then you need to unmount it. **`mount`** will show an entry beginning something like '**/dev/sda1 on /media/FLASH DRIVE...**' and you can unmount with **`sudo umount /dev/sda1`** (yes, that is umount without an '**n**'). An error will result if the device is in use, so change directory and/or close apps using files from the device. Now we can mount it just the way we want.

[DISK & DISK SPACE]

The *df* command shows you space on mounted drives: just type **df** and you'll also get a list of connected drives. It's more readable than **mount -l**, though lacking file type info. It's also quicker to type!

Finding the disk

The **/dev/sda1** refers to the first (or only) partition on **/dev/sda**. The next device plugged in will be **/dev/sdb1**. You can see what's being assigned by running **`tail -f /var/log/messages`**, then plugging in the USB device. On other Linux systems, if **/var/log/**messages draws a blank,

> "An error will result if the device is in use, so change directory and/or close apps"

try **/var/log/syslog**. Stop the tail with **CTRL+C**. Another way of seeing connected devices that aren't necessarily mounted is fdisk, a low-level tool used to divide disks up into partitions, before creating file systems on those disks (see the 'Format' boxout on page 34). Called with the list option **`sudo fdisk -l`**, it performs no partitioning, but simply lists partitions on those disks connected to your Pi. It also gives file-system information, which you need in order to mount the disk. Lastly, you need a mount point (somewhere to place the device on the file-system hierarchy) with appropriate permissions. Create one with:

```
sudo mkdir /media/usb
sudo chmod 775 /media/usb
```

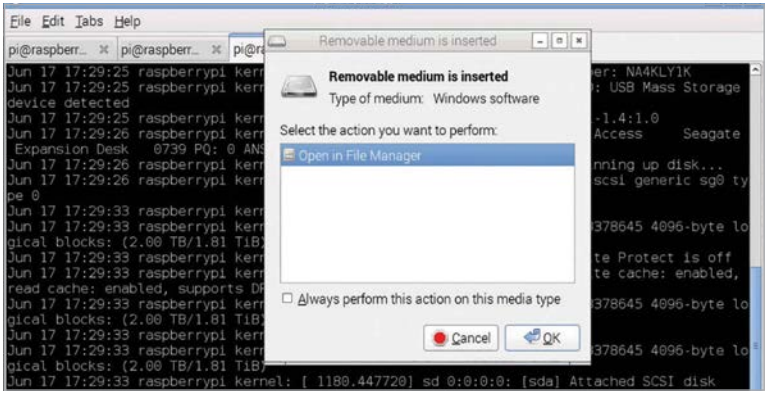

File Edit Tabs Help
Removable medium is inserted
Removable medium is inserted
Type of medium: Windows software
Select the action you want to perform:
Open in File Manager
Always perform this action on this media type
Cancel
OK
Jun 17 17:29:33 raspberrypi kernel: [1180.447720] sd 0:0:0:0: [sda] Attached SCSI disk

Fig 1 Raspbian wants to mount plugged-in disks, and take care of the details for you – note that the GUI tells you it's 'Windows software' – while the command line beneath has information for you to take control when you need the job done in a particular way, telling you it's an NTFS file system

You can then mount the disk with **`sudo mount -t vfat /dev/sda1 /media/usb`**, where VFAT (or NTFS or ext2) is the file-system type.

File-system table

Raspbian knows which disks to mount at boot time by reading the file-system table (**/etc/fstab**), and we could put our **/dev/sda1** in there, but if we start up with two drives plugged in, the wrong one may be selected. Fortunately, disks (or rather, disk partitions) have unique labels known as UUIDs randomly allocated when the partition is created. Find them with **`sudo blkid`**, which also helpfully tells you the label, if any, that often contains the make and model of external drives, or look in **/dev/disk/by-uuid**.

For an NTFS-formatted drive, we called **`sudo nano /etc/fstab`** and added the following to the end of the file:

```
/dev/disk/by-uuid/E4EE32B4EE327EBC /media/usb1t
ntfs  defaults  0   0
```

This gives the device name (yours will be different, of course), mount point, file-system type, options, and two numeric fields: the first of these should be zero (it relates to the unused dump backup program), while the second is the order of check and repair at boot: 1 for the root file system, 2 for other permanently mounted disks for data, and 0 (no check) for all others. **`man mount`** will tell you about possible options.

Editing with nano

We touched briefly on nano in chapter 2. Looking in a little more depth, the first thing to be aware of is the dozen shortcuts listed across the bottom two lines of the terminal: each is the **CTRL** key (represented by the caret **^**) held at the same time as a single letter key. For example, **^R** for ReadFile (*i.e. open*), **^O** for WriteOut (in other words, *save*), and **^X** for Exit. Remember those last two for now, and you'll be able to manage nano. However, if you learn more of them, you will really race through your editing tasks.

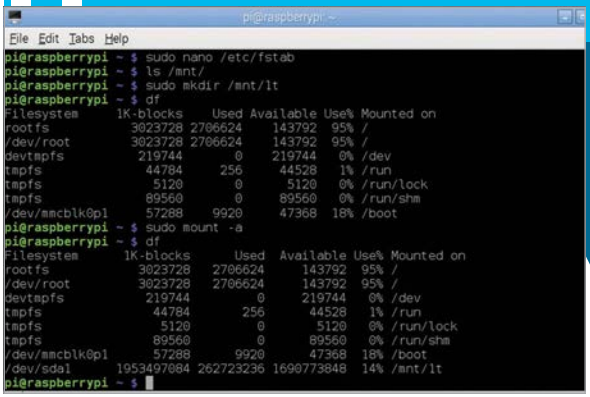

pi@raspberrypi ~
File Edit Tabs Help
pi@raspberrypi ~ $ sudo nano /etc/fstab
pi@raspberrypi ~ $ ls /mnt/
pi@raspberrypi ~ $ sudo mkdir /mnt/lt
pi@raspberrypi ~ $ df
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 3023728 2706624 143792 95% /
/dev/root 3023728 2706624 143792 95% /
devtmpfs 219744 0 219744 0% /dev
tmpfs 44784 256 44528 1% /run
tmpfs 5120 0 5120 0% /run/lock
tmpfs 89560 0 89560 0% /run/shm
/dev/mmcblk0p1 57288 9920 47368 18% /boot
pi@raspberrypi ~ $ sudo mount -a
pi@raspberrypi ~ $ df
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 3023728 2706624 143792 95% /
/dev/root 3023728 2706624 143792 95% /
devtmpfs 219744 0 219744 0% /dev
tmpfs 44784 256 44528 1% /run
tmpfs 5120 0 5120 0% /run/lock
tmpfs 89560 0 89560 0% /run/shm
/dev/mmcblk0p1 57288 9920 47368 18% /boot
/dev/sda1 1953497084 262723236 1690773848 14% /mnt/lt
pi@raspberrypi ~ $

Fig 2 Once we've put our removable disk in the file-system table (/etc/fstab), mount -a will read the config from there to mount your disks, saving you from having to remember the details

While nano lacks the power features of Emacs and Vim, its two main command-line code editor rivals, it has useful features such as a powerful Justify (**^J**), which will reassemble a paragraph of line-break strewn text into an unbroken paragraph, or apply line breaks at a fixed character length.This is a legacy of its development for email composition. **^K** cuts the line of text the cursor is on, but it isn't just a delete function: each cut is added to a clipboard. **^U** will paste the entire clipboard at the cursor position: it's great for gathering together useful snippets from a longer text.

Hit **^O** to save fstab, and the shortcut listing changes, with many now beginning M instead of **^** – this is short for Meta, which means the **ALT** key on your keyboard (once upon a time, some computers had several modifier keys, such as Super and Hyper). One 'hidden' shortcut after **^O** is that at this point, **^T** now opens a file manager to search for the file/directory where you want to save.

After saving, exit nano; now **`sudo mount -a`** will mount the external drive at the desired mount point (**Fig 2**), regardless of what else is plugged in. If you have other new entries in **/etc/fstab**, then **`sudo mount /media/usb1t`** (or whatever entry you put in fstab) will mount just that chosen device if you don't want to mount any of the others.

Having got inside connected disks, the next chapter will see us accessing all of the Pi, but remotely, from anywhere on the planet with an internet connection.

[FORMAT]

Copying a disk image negates the need to format the disk. Should you need to format a new partition, or convert a disk to ext4 format, read the manual: **man mkfs** and for individual file-system types such as **man mkfs.ext4**

The MagPi ESSENTIALS

[CHAPTER SEVEN]
PREDICTABLE NETWORKING

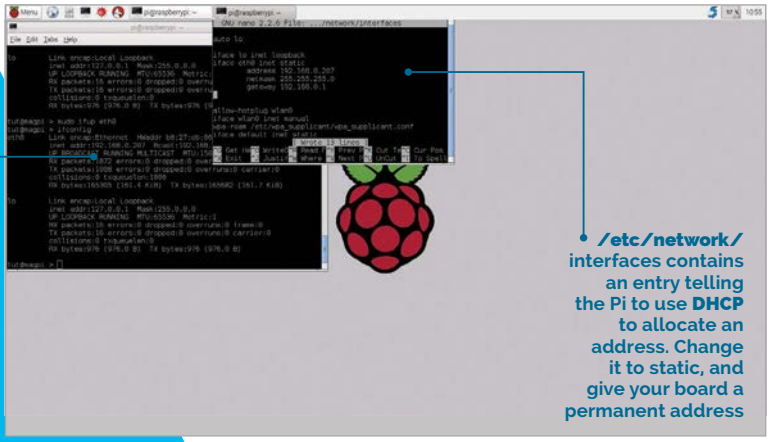
In this chapter, we give the Pi a permanent network address of its own.

They say that the network is the computer: something becoming more vital as connected devices integrate more and more real world interactivity with data and processing across the internet. Raspbian takes care of automatically connecting in most situations, but sometimes you need to override automatic configurations, to ensure a consistent network setting for your Pi project: Raspbian has the tools, and we'll show you the essentials you need to stay connected.

Plug a network cable from your **ADSL** router to your Pi and, automagically, Raspbian knows where it is on the network, and can talk to the outside world. How? Because it's setting itself up the way your router tells it to. This is thanks to **DHCP** – Dynamic Host Configuration Protocol – which provides network configuration for every device connected into a network.

Check you're using it by issuing the command: **cat /etc/network/interfaces**. This should show, amongst others, a line like: **iface etho inet dhcp** which means you're set up to receive configuration via **DHCP.** The etho is the name of your ethernet port into which you plug your network cable; a wlano here is for any wifi adaptor you may connect to your board.

For many projects on your Pi – whether something cutting edge with sensors, or something as useful but mundane as a **VPN** to protect your mobile browsing from intrusive location-based advertising – you'll need an Internet address that doesn't change, so that whatever



ifconfig gives you your Internet address; unless your Pi is sitting in a data centre, this is likely to be a private address such as in the range beginning **192.168.0.0.**

/etc/network/interfaces contains an entry telling the Pi to use **DHCP** to allocate an address. Change it to static, and give your board a permanent address

remotely connects to you can always find you. We're using local addresses here, but we touch on remotely accessing a Pi behind a router in chapter nine.

Static address

Network settings are found in **/etc/network/interfaces**; before changing them, make a back-up copy of your current working set-up with:

```
sudo cp /etc/network/interfaces /etc/network/interfaces.bak
```

Open **/etc/network/interfaces** with your favourite editor (or nano) and change **iface etho inet dhcp** to **iface etho inet static** (and the later mention of **dhcp**, under default, if you have it in your config file). Then add the following information below it, or rather, a version appropriate for your network:

```
address 192.168.0.207
netmask 255.255.255.0
gateway 192.0.0.1
```

Here, address is either the IP (version 4) address already allocated by **DHCP** to your Pi, or one with the last of the four numbers (together they're known as a dotted quad) changed to a higher number, which is unlikely to be allocated by your router's **DHCP** server. You can usually configure your router's range of addresses allocated, or reserve particular addresses.

The gateway address is the address of the device sitting between you and the Internet: your ADSL router. You can discover this with **`sudo route -n`**, the **`-n`** requesting the numeric address, rather than the name of the gateway. The address the Pi is currently using is shown with **`sudo ifconfig`**, as is netmask, which is listed as Mask.

If you look at example **/etc/network/interfaces** files online, or from other people's projects, you may also see entries for network and broadcast, showing the first three numbers of the dotted quad address of the device, followed by a 0 and a 255, respectively. These tell your computer a little more about the network that it is on, but aren't strictly necessary to getting things working. After editing the network

[IPV6]

Despite the world of connected devices crying out for the billions of addresses of IPv6, IPv4 remains the norm. Adding IPv6 to the Pi simply involves putting ipv6 on a line by itself at the end of /etc/modules.

configuration you need to restart the service:

```
sudo ifdown eth0 && sudo ifup eth0
```

Check that all is now well both with ifconfig, and trying a ping from another computer on the network.

Ping!

Ping is the most basic tool in the network testing armoury, but one which is often called upon. It sends **ICMP** (The Internet Control Message Protocol) **ECHO_REQUEST** to a device on the network. **ICMP** is built into every connected device and used for diagnostics and error messages: a ping will produce a reply from the pinged machine, which tells you it is on, and connected, and that the network is working between you and it. Information about packets lost, and time taken also helps with fault diagnosis.

A successful ping localhost from the Pi tells you not just that the local loopback interface is working, but that localhost resolves to **127.0.0.1**, the local loopback address. Name resolution is the cause of many computing problems, so we'll come back to that further down the page. Now ping the Pi from another machine on your local network: **ping 192.168.0.207**: you'll need to use the static IPv4 address you set, rather than ours, of course. If you're doing this from a Windows machine ping defaults to five attempts; from another Unix machine (another Pi, a Mac, or Ubuntu or other GNU/Linux), it will carry on until you stop it with **Ctrl-C**, unless you set a number of **ECHO_REQUEST** sends with **-c** like so: **ping -c 5 raspberrypi.org**

[COMMAND LINE WIFI]

If you add wifi to your Pi, Raspbian is ready, but the Wicd Wireless Network Connection Manager also has command line clients for both interactive and scripting use: apt-cache search wicd

What's in a name?

Numeric addresses are how connected devices see the internet, but humans prefer to remember names so **DNS** – the Domain Name System – exists to translate (resolve) addresses like raspberrypi.org into something your PC can find on the Internet, like **93.93.130.214**.

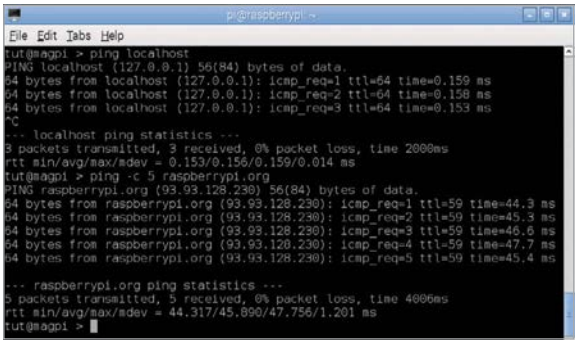

pi@raspberrypi: ~
File Edit Tabs Help
tut@magpi > ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_req=1 ttl=64 time=0.159 ms
64 bytes from localhost (127.0.0.1): icmp_req=2 ttl=64 time=0.158 ms
64 bytes from localhost (127.0.0.1): icmp_req=3 ttl=64 time=0.153 ms
^C
--- localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.153/0.156/0.159/0.014 ms
tut@magpi > ping -c 5 raspberrypi.org
PING raspberrypi.org (93.93.128.230) 56(84) bytes of data.
64 bytes from raspberrypi.org (93.93.128.230): icmp_req=1 ttl=59 time=44.3 ms
64 bytes from raspberrypi.org (93.93.128.230): icmp_req=2 ttl=59 time=45.3 ms
64 bytes from raspberrypi.org (93.93.128.230): icmp_req=3 ttl=59 time=46.6 ms
64 bytes from raspberrypi.org (93.93.128.230): icmp_req=4 ttl=59 time=47.7 ms
64 bytes from raspberrypi.org (93.93.128.230): icmp_req=5 ttl=59 time=45.4 ms
--- raspberrypi.org ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 44.317/45.890/47.756/1.201 ms
tut@magpi >

Fig 1 The humble ping is a useful diagnostic tool, telling you if a connection is working, a device is connected, and DNS is resolving

The file **/etc/hosts** contains mappings of host (computer) names to IP addresses for local network computers without publicly registered addresses: typically it will only have localhost mapping to **127.0.0.1**, unless you add further addresses. For the rest of the Internet, a **DNS** server is queried – if the address of your **ADSL** router is listed in **/etc/resolv.conf** then your router is getting **DNS** details from your **ISP** and passing them on with **DHCP** having told the Pi to ask the router.

Without **DHCP** to tell the Pi this, it's a good idea to add an entry for **DNS** servers to **/etc/network/interfaces** below the information we added earlier:

```
dns-nameservers 8.8.8.8 8.8.4.4
```

Here we've added Google's public **DNS** servers, which work reliably from anywhere. Other public DNS servers are available, some of which may guarantee not to log your IP address when you use them, or get your ISP's **DNS** server addresses from their FAQ or help website.

[FREE / PUBLIC DNS]

As well as dynamic **DNS** providers, some of those found listed at FreeDNS. com offer public **DNS** servers. For a wider listing of alternatives to **Google's DNS** servers, have a search on Google itself.

MyPiName

The line **127.0.1.1 raspberrypi** found in **/etc/hosts** is there for compatibility with some software which expects the hostname to resolve. If you changed to a static IP address, then you can change **127.0.1.1** to that new static address too.

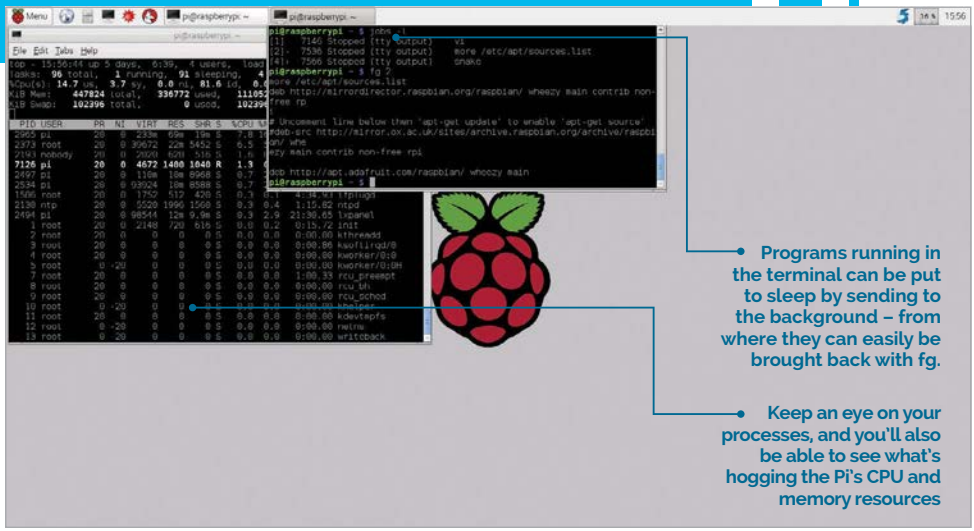
Should you wish to have your Pi change its name, then change the raspberrypi side of this entry too, as well as changing from raspberrypi in **/etc/hostname**. After this change of identity, a system restart would be a good idea. sudo reboot is essentially a pseudonym for sudo shutdown **-r 0** which will securely reboot your system from the command line. Replacing the **-r** with an **-h** will just shut the Pi down, as will sudo halt. However, **sudo /etc/init.d/hostname.sh** will tell Raspbian's Linux kernel about the new name without the need to reboot the Pi.

Lastly, to add a **.local** domain to your devices on the local network, use apt-get to install avahi-daemon for mDNS (Multicast Domain Name Servicing), using Apple's Bonjour, which works on most platforms. You should now be able to connect to **<yournewPiname>**. local from other Bonjour-enabled computers on your network without even needing to use the static IP address you allocated.

The MagPi ESSENTIALS

[CHAPTER EIGHT]
COMMAND LINE PI

As close to perfect as Raspbian is, things can go wrong . In this chapter, we learn that there's no need to turn the Raspberry Pi off and on again: just kill the process!


Programs running in the terminal can be put to sleep by sending to the background – from where they can easily be brought back with fg.
Keep an eye on your processes, and you'll also be able to see what's hogging the Pi's CPU and memory resources

Ever lost the 'off switch' for a program? Sometimes a piece of software you're running seems to have no inclination to stop: either you cannot find how to quit, or the app has a problem, and won't respond to your q, **Ctrl-C**, or whatever command should close it down.

There's no need to panic, and certainly no need to reboot: just identify the process and quietly kill it. We'll show you how, and look at what else can be done with knowledge of processes.

Processes

Find the many processes running on your Pi with the **ps** command. As a minimum, on Raspbian, it's usually called with the **a** and **x** switches – which together give all processes, rather than just those belonging to a user, and attached to a **tty** – and with **u** to see processes by user. **w** adds wider output, and ww will wrap over the line end to display information without truncating.

Type **ps auxww** to see, then try with just **a** or other combinations. You will notice that these options work without the leading dash seen for other commands. Both the lack of dashes, and the particular letters, **a** and **x**, date back to the original Unix ps of the early 1970s, maintained through various revisions by one of Unix's two family

branches, **BSD**, and baked into the first GNU/Linux ps. Unix's other branch, System V, had extended and changed ps with new options and new abbreviations for command switches, so for ps ax you may see elsewhere ps -e (or -ef or -ely to show in long format).

The ps aux listing has various headers, including the **USER** which owns the process, and the **PID**, or Process IDentification number. This starts with 1 for init, the parent process of everything that happens in userspace after the Linux kernel starts up when you switch the Pi on.

Knowing the **PID** makes it easy to kill a process, should that be the easiest way of shutting it down. For example, to kill a program with a **PID** of 3012, simply enter **kill 3012**, and to quickly find the process in the first place, use grep on the ps list. For example, locating vi processes:

```
ps aux | grep -i  vi
```

The -i (ignore case) isn't usually necessary, but occasionally a program may break convention, and contain upper case letters in its filename. You can also use killall to kill by program name: **killall firefox**

Piping commands

Naturally you can pipe ps's output to select the PID and feed directly to the kill command:

```
kill $(ps aux | grep '[f]irefox' | awk '{print $2}')
```

We don't have space for an in-depth look at awk (we're using it here to print the second field of grep's output: the **PID**), but the **[f]** trick at the beginning of Firefox (or whatever named process you want to kill) singles out the Firefox process; in the vi example above, grep found the grep process itself as well as vi (and anything with the letter sequence vi in its name).

The output of ps also shows you useful information like percentage of memory and CPU time used, but it's more useful to see these changing in real time. For this, use top, which also shows total CPU and memory use in the header lines, the latter in the format that you can also find by issuing the command free. For an improved top:

[KEEP ON TOP]

Using a virtual console, it can be worth keeping htop running so that if there are any problems you can Ctrl-Alt-Fn there for a quick look for any problems – even if the GUI freezes

[QUICKER BOOT]

The start-up process of Raspbian Wheezy is controlled by SysVinit, but like other GNU/Linux distributions will eventually change to the new, faster SystemD. This will change start up processes, but instructions here will still be relevant

```
apt-get install htop
```

htop is scrollable, both horizontally and vertically, and allows you to issue commands (such as **k** for kill) to highlighted processes. When you've finished, both top and htop are exited with **q**, although in htop you may care to practice by highlighting the htop process and killing it from there (see Fig 1). htop also shows load over the separate cores of the processor if you have a Pi 2.

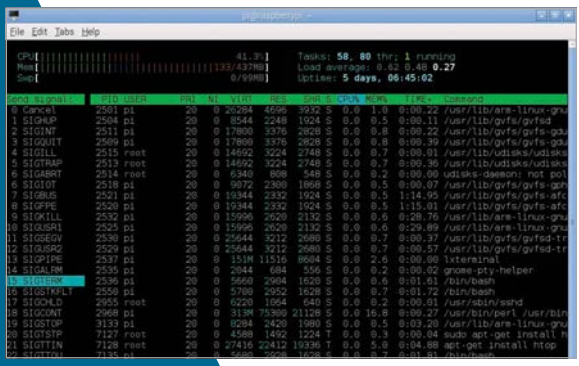
Background job

Placing an ampersand after a command in the shell, places the program in the background – try with: man top & and you'll get an output like: **[1] 12768**

The first number is a job number, assigned by the shell, and the second the PID we've been working with above. man top is now running in the background, and you can use the job control number to work with the process in the shell. Start some other processes in the background if you wish (by appending &), then bring the first – man top – to the foreground with **fg 1**. Now you should see man running again.

You can place a running shell program in the background by 'suspending' it with **Ctrl-z**. fg will always bring back the most recently suspended or backgrounded job, unless a job number is specified. Note that these job numbers apply only within the shell where the process started. Type jobs to see background processes; jobs **-l** adds in Process IDs (PID) to the listing.

Fig 1 htop tells you what's running, what resources it's using, and lets you interact with the process, even killing htop from within htop



Signals

When we sent a kill signal from htop, we were given a choice of signal to send. The most important are **SIGTERM**, **SIGINT** and **SIGKILL**. The first was the default when we killed from htop, and is the signal kill sends if not called with a modifier: it tells a process to stop, and most programs will respond by catching

the signal, and first saving any data they need to save and releasing system resources before quitting.

kill -2 sends **SIGINT** which is equivalent to stopping a program from the terminal with **Ctrl-C**: you could lose data. Most drastic is **kill -9** to send **SIGKILL**, telling the kernel to let the process go with no warning. Save this one for when nothing else works.

Mildest of all is the Hang Up (**HUP**) signal, called with **kill -1**, which many daemons are programmed to treat as a call to simply re-read their configuration files and carry on running. It's certainly the safest signal to send on a critical machine.

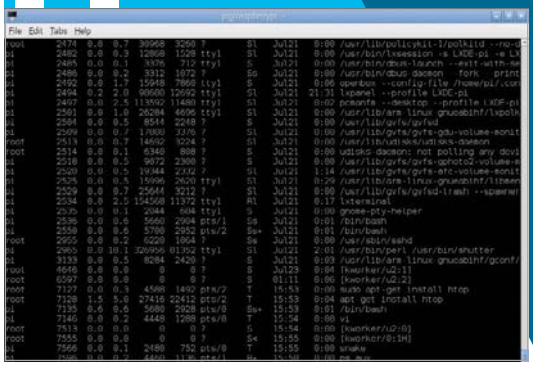


Fig 2 Everything running has a process ID (PID), that can be used to control that program, find them all with ps aux.

Staying on

nohup will run a program which will continue after the terminal from which it is started has closed, ignoring the consequent **SIGHUP** (**hangup**) signal. As the process is detached from the terminal, error messages and output are sent to the file **nohup.out** in whichever directory you were in when you started the process. You can redirect it – as we did in part 4 – with **1>** for **stdout** and **2>** for **stderr**; **&>** is a special case for redirecting both **stdout** and **stderr**:

```
nohup myprog &>backgroundoutput.txt &
```

One use of **NOHUP** for Pi users is to be able to set something in motion from a **SSH** session, that will continue after an interruption to that session. For example restarting the network connection to which you are connected:

```
sudo nohup sh -c "ifdown eth0 && ifup eth0"
```

Note that the nohup.out log file created here will need sudo priveleges to read – or reassign with:

```
sudo chown pi:pi nohup.out
```

[KEEP ON RUNNING]

nohup is useful for a program that will be running for some time in the background – perhaps a sensor project you are working on – until you feel happy enough to add it to Raspbian's start up processes

The MagPi ESSENTIALS

[CHAPTER NINE]
REMOTE PI

In this chapter, we cover accessing the Pi from remote PCs and devices with Secure Shell.

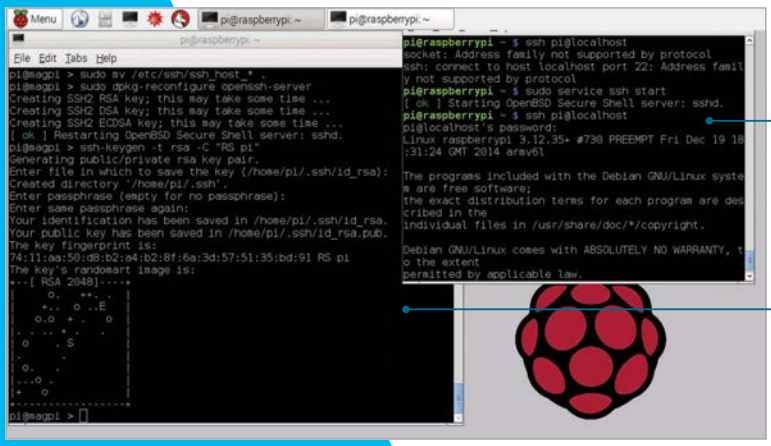
It's great that the Raspberry Pi is so portable, but sometimes you may want to use it without taking it with you. Here, the Pi's default operating system is a real strength, as Unix-like operating systems have been used this way for over 40 years.

Over time, as the internet has given the opportunity for malicious users to connect to your computers, old standards like telnet and rlogin have been replaced by Secure Shell (SSH), based on public-key cryptography. The good news is that once set up, secure connections are just as easy, and are open to scripted, automatic connection for your projects.

With Raspbian, the SSH server is enabled by default. If not, run **`sudo raspi-config`**, then enable SSH (found in the advanced settings). Check the IP address assigned to the Pi with **`ifconfig`** ('inet addr' for the etho interface). Now you can try connecting from another computer on your network.

Connecting with SSH

From a Mac or GNU/Linux computer, just use **`ssh`** from a terminal to connect to your Pi. Assuming a default setup, and ifconfig revealing an IP address of 192.168.0.2, connect with **`ssh pi@192.168.0.2`** and enter your password when asked. From a Windows PC, you'll need to install an SSH client: we recommend Putty (**bit.ly/1dTlX8g**), which also works with SCP, telnet, and rlogin, and can connect to a serial port. Android users can try the ConnectBot client.


Menu
pi@raspberrypi: ~
File Edit Tabs Help
pi@magpi > sudo mv /etc/ssh/ssh_host_* .
pi@magpi > sudo dpkg-reconfigure openssh-server
Creating SSH2 RSA key; this may take some time ...
Creating SSH2 DSA key; this may take some time ...
Creating SSH2 ECDSA key; this may take some time ...
[ok] Restarting OpenBSD Secure Shell server: sshd.
pi@magpi > ssh-keygen -t rsa -C "RS pi"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pi/.ssh/id_rsa):
Created directory '/home/pi/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pi/.ssh/id_rsa.
Your public key has been saved in /home/pi/.ssh/id_rsa.pub.
The key fingerprint is:
74:11:aa:50:d8:b2:a4:b2:8f:6a:3d:57:51:35:bd:91 RS pi
The key's randomart image is:
+---[RSA 2048]----+
pi@magpi >
pi@raspberrypi ~ $ ssh pi@localhost
Socket: Address family not supported by protocol
ssh: connect to host localhost port 22: Address famil
y not supported by protocol
pi@raspberrypi ~ $ sudo service ssh start
[ok] Starting OpenBSD Secure Shell server: sshd.
pi@raspberrypi ~ $ ssh pi@localhost
pi@localhost's password:
Linux raspberrypi 3.12.35+ #730 PREEMPT Fri Dec 19 18
:31:24 GMT 2014 armv6l
The programs included with the Debian GNU/Linux syste
m are free software;
the exact distribution terms for each program are des
cribed in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, t
o the extent
permitted by applicable law.

You can test on the Pi if SSH is running, and start the service from the command line – as you can any service (look in `/etc/init.d/` and `/etc/init/` if you're curious about other services)

The Raspbian install image shares its keys with everyone else who has a copy. Generate your own, and personal keys for the user account, for secure remote access

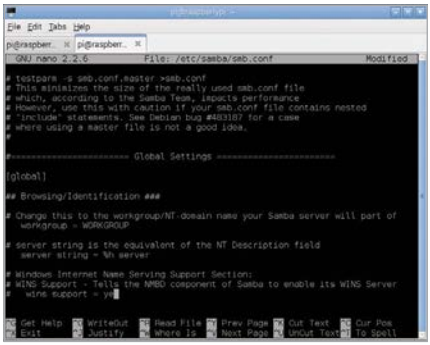

File Edit Tabs Help
GNU nano 2.2.6 File: /etc/samba/smb.conf Modified
testparm -s smb.conf.master >smb.conf
This minimizes the size of the really used smb.conf file
which, according to the Samba Team, impacts performance
However, use this with caution if your smb.conf file contains nested
"include" statements. See Debian bug #483187 for a case
where using a master file is not a good idea.
#
#======================= Global Settings =======================
[global]
Browsing/Identification
# Change this to the workgroup/NT-domain name your Samba server will part of
workgroup = WORKGROUP
# server string is the equivalent of the NT Description field
server string = %h server
Windows Internet Name Serving Support Section:
WINS Support - Tells the NMBD component of Samba to enable its WINS Server
wins support = yes
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Fig 1 There's a lot of configuration in Samba, but simply adding your WORKGROUP name to the default settings should get you up and running

You should now be at the command-line interface of your Pi. If you got any sort of error, check from the Pi that SSH is really up and running by entering **ssh@localhost** on the Pi itself. If that works, SSH is up and running on the Pi, so take a closer look at network settings at both ends.

Hello, World

Now we can access the Pi on the local network, it's time to share with the world. Step one, before even thinking about going further, change the **PermitRootLogin yes** entry in **/etc/ssh/sshd_config** to read: **PermitRootLogin no** using **sudo nano**. After making any changes to the SSH server's configuration, you must restart the service for them to take effect, or at least reload the configuration file: **sudo service ssh reload**. Note there's also a file in **/etc/ssh/** called **ssh_config**, which is for the SSH client; the **d** in **sshd_config** is short for 'daemon', the Unix term for a service which runs constantly in the background.

You can also change port 22 to any unlikely number, but be sure to check it still works. You'll need to begin **ssh -p 12123** (or whichever port you have chosen) to tell your client you're not using the default port 22.

To reach your Pi from anywhere on the internet, you need an IP address, which will connect you to your board even though it's behind an ADSL router. Of course, if your Pi is in a data centre, with its own public IP address, you don't need any workaround.

There are numerous services such as DuckDNS.org providing free-of-charge dynamic DNS (DDNS), to point a constant IP address to the changing one allocated to you by your ISP. However, the largest of these, DynDNS, has ended its free service, which provides a useful reminder that you cannot assume that a free service will be around for ever.

There are several steps to configuring a DDNS setup, no matter which service and software client you choose. Some are detailed in the **raspberrypi.org** forums, and there's a good guide to ddclient at **samhobbs.co.uk**.

Otherwise, if your broadband router can handle both port forwarding and dynamic DNS, you can set it up to point to port 22 (or a chosen alternate port) on the Pi. You may even find your ISP offers static IP addresses.

[INTERRUPTED SERVICE]

While you can restart most services with **sudo service ssh restart**, replacing **restart** with **reload** permits configuration changes to be registered with less disruption, which is key for some projects.

Bye bye FTP

FTP, dating back to 1971, was not designed for security: data, and even passwords, are transmitted unencrypted. The Secure Copy Program (SCP), which runs over SSH, is included in the Pi's SSH packages. The syntax of the command is familiar, as it mimics the command-line cp program, adding in the path for the remote side of the transaction's network address `scp pi@192.168.0.2:/home/pi/bin/lein`.

Here we're transferring a file from the Pi, across a local network, to the current location (the dot shortcut). Note that you can use wildcards for groups of similarly named files, and can recursively copy directories and their contents with the `-r` switch after `scp`.

A secure key

If you're trying this on something other than Raspbian, you may not have the SSH server installed. It's often found in a package called **openssh-server**. With Raspbian, you have a pair of keys (public and private) in **/etc/ssh/**. Unfortunately, they'll be the same as those held by everyone else with a copy of the Raspbian image that you downloaded. Follow these steps to create new keys.

First, remove the existing keys:

```
sudo rm /etc/ssh/ssh_host_*
```

Alternatively, you can move them somewhere out of the way. Regenerate the system-wide keys with:

```
sudo dpkg-reconfigure openssh-server
```

For keys personal to you as a user, type `ssh-keygen -t rsa -C "comment"`, where "comment" is anything you want to identify the key with: name, email, or machine and project, for example. If you press **ENTER** at the passphrase step, you'll get a key with no passphrase, which makes life easier when making scripted (automated) connections, but removes an extra layer of security. You can create keys from any computer with the SSH package, and move the public key to the Pi, but we'll work on the assumption that the Pi is the only handy Unix-like computer, and we'll be generating the keys there.

[SAMBA STEPS]

Samba is *extremely* well documented, with separate man pages for everything from **smb.conf** to **smbpasswd**, and excellent online books at **samba.org** – look for smb.conf examples.

If you accepted the defaults, your personal keys will now be in **~/.ssh** with the correct permissions. By default, sshd looks in **~/.ssh/authorized_keys** for public keys, so copy the contents of **id_rsa.pub** to there. The following will work even if you already have an **authorized_keys** with contents (make sure you use both **>>** symbols with no gap between them):

```
cd ~/.ssh && cat id_rsa.pub
 >>  authorized_keys
```

Using SCP, copy the private key to **~/.ssh** on your laptop, or wherever you will access the Pi from, removing it from the Pi if it's to act as the server. Once you confirm SSH works without passwords, you can edit **/etc/ssh/sshd_config** to include **PasswordAuthentication no**. If you are sticking with passwords, replace 'raspberry' with something stronger.

[LOST KEYS?]

The private key half of your key pair should be kept secure – but safe, too. Keep a backup of the private key on a memory card in a safe place.

Shared drive

You may be using a service like Dropbox to share files between machines. There is no need to do this on a local network, as putting Samba on the Pi lets even Windows PCs see it in Network Neighbourhood, and access it as a shared drive:

```
sudo apt-get install samba samba-common-bin
```

Now edit **/etc/samba/smb.conf** with a WORKGROUP value (for Windows XP and earlier; try **workgroup = WORKGROUP**) and/or HOME (For Windows 7 and above). Ensure that Samba knows **pi** is a network user:

```
sudo smbpasswd -a pi
```

Then restart with:

```
sudo service samba restart
```

The Pi should now show up in Windows Explorer under Network. You can fine-tune **smb.conf** for what's shared (including printers), and permissions.

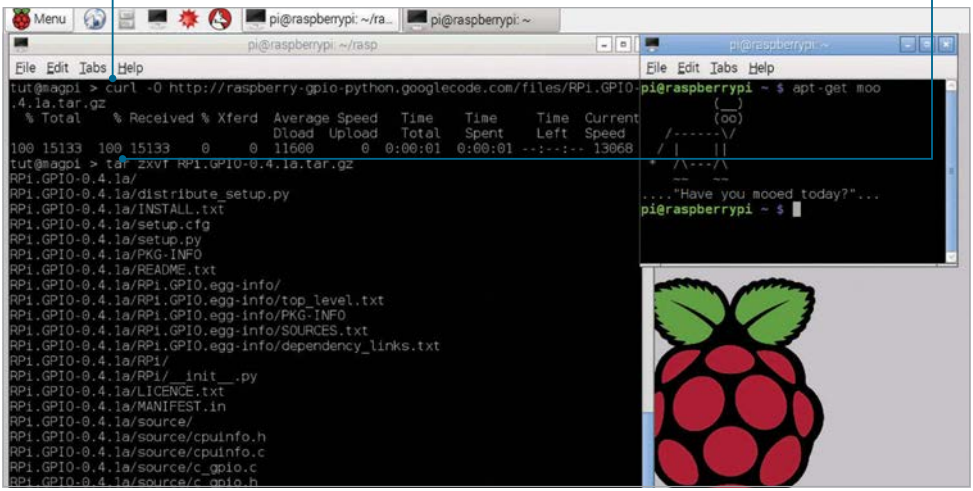
The MagPi ESSENTIALS

[CHAPTER TEN]
DOWNLOADING & INSTALLING

In this chapter, we look at downloading and unpacking software, and learn how to create new Raspbian SD cards.

curl can be used in place of **wget** for simply downloading files, but its strengths lie elsewhere, in its extensive features – these take in everything from proxy support and user authentication, to FTP upload and cookies

The **tar** command packs or unpacks an archive of files and directories; it also handles uncompressing the download first



Running an **apt** command (see chapter 3) allows access to a huge collection of software – several thousands of packages in the main Raspbian repository – but sometimes we need to add software from outside the main repository.

If we are lucky, we find that someone has packaged up the software in the .deb format used by Raspbian, or even created a whole repository to take care of the dependencies. We'll look briefly both at adding repositories, and dealing with other kinds of downloads, trying the venerable vi editor along the way.

Information about repositories is kept in the **/etc/apt/sources.list** file, which on a new install just contains the Raspbian repository. There are actually two of these repositories: one for the binaries you use, and one to get the source code, which enables you to learn from or modify any Raspbian software. To add a new repository, edit the file and add it in the same format:

```
deb http://apt.adafruit.com/raspbian/ wheezy main
```

Wheezy is a Debian release name: all Debian releases have been named after characters in the *Toy Story* series of films since 1996 (former Debian project leader Bruce Perens was involved in the early development of Debian while working at Pixar). Jessie followed Wheezy in April 2015.

Most software is in the main repository. Other components, like non-free, allow repositories to contain software you may not be free to pass on, keeping it separate from Raspbian's FOSS repository. Main can be freely copied or mirrored anywhere.

[VI IMPROVED]

If you really want to get to grips with vim, you'll need to **apt-get install vim** – the vim.tiny package already in Raspbian is very limited.

vi editor

If you tried **robots** after chapter 3, you'll be used to the **hjkl** keys for directional movement. If you're feeling brave, now is the chance to try them in a serious task: editing the **sources.list** file. It's not compulsory – use nano if you wish – but sooner or later you could come across a Linux or BSD computer without nano; vi is always the default on such machines. It's there in Raspbian in the form of vim.tiny, which you can call as **vi**.

```
vi /etc/apt/sources.list
```

vi is a modal editor: you start in command mode, moving to the line you wish to modify (**hjkl** in place of arrow keys); to edit, hit **i** for insert mode, and you can now edit the line under the cursor. The **ESC** key gets you back to command mode. If editing has gone okay, then **ESC** followed by **:wq** will get you out of vi; if not, **:q!** will leave without saving.

There's a lot more to vi: in the form of the more powerful vim (vi improved), it's a popular editor in the Ruby community. For now, be happy that when faced with a lack of nano, you'll get by: there's an old joke of someone running vi for years, not because they liked it, but because they couldn't figure out how to exit it!

wget & curl

Having added our repository to **sources.list**, we need to get the key and use **apt-key** to install it. Packages authenticated using keys added by **apt-key** will be considered trusted.

```
wget -O - -q https://apt.adafruit.com/apt.adafruit.
com.gpg.key | apt-key add -
```

Wget downloads from the URL given. The **-O** directs the download to stdout, from where it is piped to apt-key (the trailing dash there tells apt-key to read its input from the stdin stream, which is where it receives the output from wget). After any change to the sources.list file, you must run:

```
sudo apt-get update
```

This updates Raspbian's knowledge of what's available to install from Adafruit's packages: see **apt.adafruit.com** for details, e.g. **apt-get install wiringpi**.

Wget is a simple but robust download tool, with a powerful recursive feature that helps fetch entire websites, but it does have mild security risks, so be careful using it to fetch scripts. curl is a file transfer tool, working with many protocols, that can be used for simple downloads. It dumps to stdout by default; to save as a file with the same name as the resource in the URL, use the **-O** switch:

```
curl -O http://raspberry-gpio-python.googlecode.com/files/RPi.GPIO-0.4.1a.tar.gz
```

Unzip

The Python GPIO library downloaded above is compressed with gzip, which losslessly reduces the size of files, and can be decompressed with **gunzip**. The contents here are files rolled into a tar archive (instead of **.tar.gz**, you'll sometimes find similar archives ending **.tgz**), and the **tar** command can do the decompression and untarring in one:

```
tar zxvf Rpi.GPIO-0.4.1a.tar.gz
```

Note that the dash is not needed for single letter options in tar. The first switch, **z**, calls gzip to decompress the archive, then **x** extracts the contents. **v** is verbose, informing you of the process as it happens, and **f** tells tar to work with the named file(s), rather than stdin. Miss out the **z** and tar should automatically detect the necessary compression operation.

The result in this case is a folder containing, among other things, a setup script to run the installation (read the **INSTALL.txt** file first):

```
cd RPi.GPIO-0.4.1a
sudo python setup.py install
```

While gzip is more efficient than zip (and even more efficient options like bzip2 are available), sometimes you'll get a plain old zip file, in which case **unzip** is the command you want.

```
unzip 2014-12-24-wheezy-raspbian.zip
```

Disk image

Having downloaded and unzipped an image for Raspbian, you cannot copy it across to a microSD card with regular **cp**, which would simply put a copy of it as a file on the card. We need something to replace the SD card's file system with the file system and contents that exist inside the Raspbian disk image, byte-for-byte, and for this we can use a handy little built-in utility called **dd**.

dd converts and copies files – any files, even special devices like /dev/zero or /dev/random (you can make a file full of zeroes or random noise) – precisely copying blocks of bytes. To copy our Raspbian image, we need to unmount the SD card we've plugged in. Use **sudo fdisk -l** both before and after plugging in the SD card (you can also use **df** to see what's mounted). If, say, a /dev/sdb appears, with the size equal to the SD card, then unmount with **umount /dev/sdb1**. Now copy the disk image with:

```
sudo dd if=~/Downloads/2015-05-05-wheezy-raspbian.img
of=/dev/sdb bs=1M
```

Development of Raspbian's ancestor Unix started in 1969, so we've covered a few utilities with a long heritage in this book, but that **if=...** in place of the usual dashes for command-line options indicates a lineage stretching back to the early 1960s, and IBM's Data Definition (DD) statement from the OS/360 Job Control Language (JCL).

Be very careful that the destination matches the correct disk, or you will lose the contents of another storage device! The **bs=1M** is a block size default; **4M** would be another safe option. Now put the card in another Pi and go and have fun!

[EASTER EGG]

Read **man apt**, and you may see: "This APT has Super Cow Powers." If it's there, try typing **apt-get moo** to see what happens.

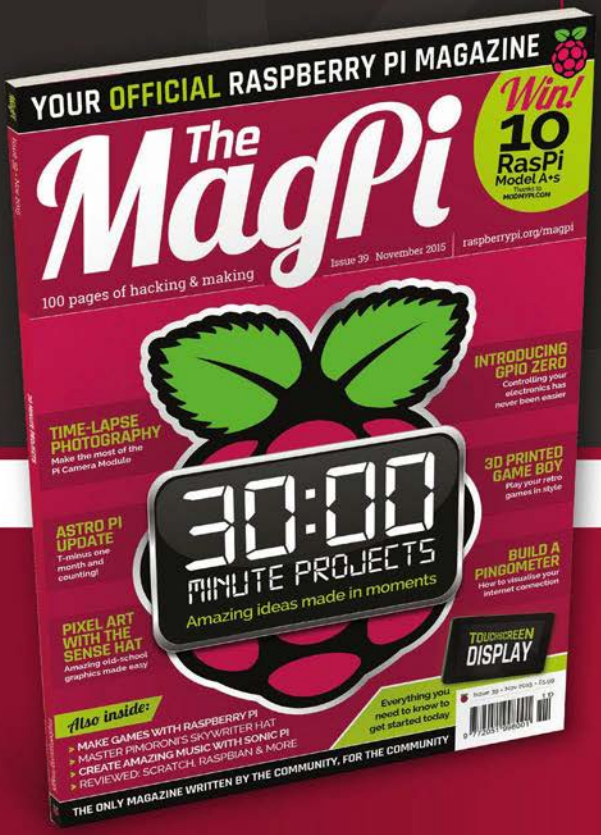
SUBSCRIBE **TODAY!**

& **SAVE** UP TO **25%**

Subscribe to the Official Raspberry Pi mag today for a whole host of benefits

Subscription benefits

- Save up to 25% on the price
- Free delivery to your door
- Never miss a single issue


YOUR OFFICIAL RASPBERRY PI MAGAZINE
The MagPi
Win! 10 RasPi Model A+s
100 pages of hacking & making
Issue 39 November 2015
raspberrypi.org/magpi
INTRODUCING GPIO ZERO
Controlling your electronics has never been easier
TIME-LAPSE PHOTOGRAPHY
Make the most of the Pi Camera Module
3D PRINTED GAME BOY
Play your retro games in style
ASTRO PI UPDATE
T-minus one month and counting!
30:00 MINUTE PROJECTS
Amazing ideas made in moments
BUILD A PINGOMETER
How to visualise your internet connection
PIXEL ART WITH THE SENSE HAT
Amazing old-school graphics made easy
TOUCHSCREEN DISPLAY
Also inside:
MAKE GAMES WITH RASPBERRY PI
MASTER PIMORONI'S SKYWRITER HAT
CREATE AMAZING MUSIC WITH SONIC PI
REVIEWED: SCRATCH, RASPBIAN & MORE
Everything you need to know to get started today
THE ONLY MAGAZINE WRITTEN BY THE COMMUNITY, FOR THE COMMUNITY

Pricing

Get the first six issues:

£30 (UK)
£45 (EU)
$69 (US)
£50 (Rest of World)

Subscribe for a year:

£55 (UK)
£80 (EU)
$129 (US)
£90 (Rest of World)

Direct Debit
£12.99 (UK) **(quarterly)**

NEW US SUBS OFFER!
imsnews.com/magpi

How to subscribe:

- **bit.ly/MagPiSubs (UK – ROW)**
- Local call +44 (0)1202 586848
- **imsnews.com/magpi (US)**
- Call toll free 800-428-3003 **(US)**

Available on the App Store

Get it on Google play

raspberrypi.org/**magpi**